JavaScript Classes

In JavaScript, classes are the special type of functions. We can define the class just like function declarations and function expressions.

The JavaScript class contains various class members within a body including methods or constructor. The class is executed in strict mode. So, the code containing the silent error or mistake throws an error.

The class syntax contains two components:

* Class declarations
* Class expressions

Class Declarations

A class can be defined by using a class declaration. A class keyword is used to declare a class with any particular name. According to JavaScript naming conventions, the name of the class always starts with an uppercase letter.

Class Declarations Example

Let's see a simple example of declaring the class.

1. **<script>**
2. //Declaring class
3. class Employee
4. {
5. //Initializing an object
6. constructor(id,name)
7. {
8. this.id=id;
9. this.name=name;
10. }
11. //Declaring method
12. detail()
13. {
14. document.writeln(this.id+" "+this.name+"**<br>**")
15. }
16. }
17. //passing object to a variable
18. var e1=new Employee(101,"Martin Roy");
19. var e2=new Employee(102,"Duke William");
20. e1.detail(); //calling method
21. e2.detail();
22. **</script>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=JavaScriptClassDeclarationsExample)

**Output:**

101 Martin Roy

102 Duke William

Class Declarations Example: Hoisting

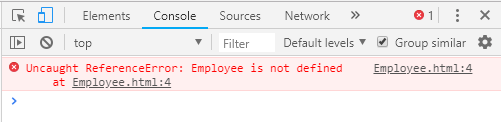
Unlike function declaration, the class declaration is not a part of JavaScript hoisting. So, it is required to declare the class before invoking it.

Let's see an example.

1. **<script>**
2. //Here, we are invoking the class before declaring it.
3. var e1=new Employee(101,"Martin Roy");
4. var e2=new Employee(102,"Duke William");
5. e1.detail(); //calling method
6. e2.detail();
8. //Declaring class
9. class Employee
10. {
11. //Initializing an object
12. constructor(id,name)
13. {
14. this.id=id;
15. this.name=name;
16. }
17. detail()
18. {
19. document.writeln(this.id+" "+this.name+"**<br>**")
20. }
21. }
22. **</script>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=JavaScriptClassDeclarationsHoistingExample)

**Output:**



Class Declarations Example: Re-declaring Class

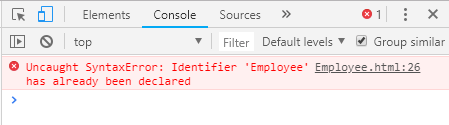
A class can be declared once only. If we try to declare class more than one time, it throws an error.

Let's see an example.

1. **<script>**
2. //Declaring class
3. class Employee
4. {
5. //Initializing an object
6. constructor(id,name)
7. {
8. this.id=id;
9. this.name=name;
10. }
11. detail()
12. {
13. document.writeln(this.id+" "+this.name+"**<br>**")
14. }
15. }
16. //passing object to a variable
17. var e1=new Employee(101,"Martin Roy");
18. var e2=new Employee(102,"Duke William");
19. e1.detail(); //calling method
20. e2.detail();
21. //Re-declaring class
22. class Employee
23. {
24. }
25. **</script>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=JavaScriptClassDeclarationsRedeclaringClassExample)

**Output:**



Class expressions

Another way to define a class is by using a class expression. Here, it is not mandatory to assign the name of the class. So, the class expression can be named or unnamed. The class expression allows us to fetch the class name. However, this will not be possible with class declaration.

Unnamed Class Expression

The class can be expressed without assigning any name to it.

Let's see an example.

1. **<script>**
2. var emp = class {
3. constructor(id, name) {
4. this.id = id;
5. this.name = name;
6. }
7. };
8. document.writeln(emp.name);
9. **</script>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=JavaScriptUnnamedClassExpressionExample)

**Output:**

emp

Class Expression Example: Re-declaring Class

Unlike class declaration, the class expression allows us to re-declare the same class. So, if we try to declare the class more than one time, it throws an error.

1. **<script>**
2. //Declaring class
3. var emp=class
4. {
5. //Initializing an object
6. constructor(id,name)
7. {
8. this.id=id;
9. this.name=name;
10. }
11. //Declaring method
12. detail()
13. {
14. document.writeln(this.id+" "+this.name+"**<br>**")
15. }
16. }
17. //passing object to a variable
18. var e1=new emp(101,"Martin Roy");
19. var e2=new emp(102,"Duke William");
20. e1.detail(); //calling method
21. e2.detail();
23. //Re-declaring class
24. var emp=class
25. {
26. //Initializing an object
27. constructor(id,name)
28. {
29. this.id=id;
30. this.name=name;
31. }
32. detail()
33. {
34. document.writeln(this.id+" "+this.name+"**<br>**")
35. }
36. }
37. //passing object to a variable
38. var e1=new emp(103,"James Bella");
39. var e2=new emp(104,"Nick Johnson");
40. e1.detail(); //calling method
41. e2.detail();
42. **</script>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=JavaScriptClassExpressionRedeclaringClassExample)

**Output:**

101 Martin Roy

102 Duke William

103 James Bella

104 Nick Johnson

Named Class Expression Example

We can express the class with the particular name. Here, the scope of the class name is up to the class body. The class is retrieved using class.name property.

1. **<script>**
2. var emp = class Employee {
3. constructor(id, name) {
4. this.id = id;
5. this.name = name;
6. }
7. };
8. document.writeln(emp.name);
9. /\*document.writeln(Employee.name);
10. Error occurs on console:
11. "ReferenceError: Employee is not defined
12. \*/
13. **</script>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=JavaScriptNamedClassExpressionExample)

**Output:**

Employee

# JavaScript Objects

A javaScript object is an entity having state and behavior (properties and method). For example: car, pen, bike, chair, glass, keyboard, monitor etc.

JavaScript is an object-based language. Everything is an object in JavaScript.

JavaScript is template based not class based. Here, we don't create class to get the object. But, we direct create objects.

## Creating Objects in JavaScript

There are 3 ways to create objects.

Play Video

1. By object literal
2. By creating instance of Object directly (using new keyword)
3. By using an object constructor (using new keyword)

## 1) JavaScript Object by object literal

The syntax of creating object using object literal is given below:

1. object={property1:value1,property2:value2.....propertyN:valueN}

As you can see, property and value is separated by : (colon).

Let’s see the simple example of creating object in JavaScript.

1. **<script>**
2. emp={id:102,name:"Shyam Kumar",salary:40000}
3. document.write(emp.id+" "+emp.name+" "+emp.salary);
4. **</script>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=object1js)

#### **Output of the above example**

102 Shyam Kumar 40000

## 2) By creating instance of Object

The syntax of creating object directly is given below:

1. var objectname=new Object();

Here, **new keyword** is used to create object.

Let’s see the example of creating object directly.

1. **<script>**
2. var emp=new Object();
3. emp.id=101;
4. emp.name="Ravi Malik";
5. emp.salary=50000;
6. document.write(emp.id+" "+emp.name+" "+emp.salary);
7. **</script>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=object2js)

#### **Output of the above example**

101 Ravi 50000

## 3) By using an Object constructor

Here, you need to create function with arguments. Each argument value can be assigned in the current object by using this keyword.

The **this keyword** refers to the current object.

The example of creating object by object constructor is given below.

1. **<script>**
2. function emp(id,name,salary){
3. this.id=id;
4. this.name=name;
5. this.salary=salary;
6. }
7. e=new emp(103,"Vimal Jaiswal",30000);
9. document.write(e.id+" "+e.name+" "+e.salary);
10. **</script>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=object3js)

#### **Output of the above example**

103 Vimal Jaiswal 30000

## Defining method in JavaScript Object

We can define method in JavaScript object. But before defining method, we need to add property in the function with same name as method.

The example of defining method in object is given below.

1. **<script>**
2. function emp(id,name,salary){
3. this.id=id;
4. this.name=name;
5. this.salary=salary;
7. this.changeSalary=changeSalary;
8. function changeSalary(otherSalary){
9. this.salary=otherSalary;
10. }
11. }
12. e=new emp(103,"Sonoo Jaiswal",30000);
13. document.write(e.id+" "+e.name+" "+e.salary);
14. e.changeSalary(45000);
15. document.write("**<br>**"+e.id+" "+e.name+" "+e.salary);
16. **</script>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=object4js)

#### **Output of the above example**

103 Sonoo Jaiswal 30000  
103 Sonoo Jaiswal 45000

## JavaScript Object Methods

The various methods of Object are as follows:

|  |  |  |
| --- | --- | --- |
| **S.No** | **Methods** | **Description** |
| 1 | [Object.assign()](https://www.javatpoint.com/javascript-object-assign-method) | This method is used to copy enumerable and own properties from a source object to a target object |
| 2 | [Object.create()](https://www.javatpoint.com/javascript-object-create-method) | This method is used to create a new object with the specified prototype object and properties. |
| 3 | [Object.defineProperty()](https://www.javatpoint.com/javascript-object-defineproperty-method) | This method is used to describe some behavioral attributes of the property. |
| 4 | [Object.defineProperties()](https://www.javatpoint.com/javascript-object-defineproperties-method) | This method is used to create or configure multiple object properties. |
| 5 | [Object.entries()](https://www.javatpoint.com/javascript-object-entries-method) | This method returns an array with arrays of the key, value pairs. |
| 6 | [Object.freeze()](https://www.javatpoint.com/javascript-object-freeze-method) | This method prevents existing properties from being removed. |
| 7 | [Object.getOwnPropertyDescriptor()](https://www.javatpoint.com/javascript-object-getownpropertydescriptor-method) | This method returns a property descriptor for the specified property of the specified object. |
| 8 | [Object.getOwnPropertyDescriptors()](https://www.javatpoint.com/javascript-object-getownpropertydescriptors-method) | This method returns all own property descriptors of a given object. |
| 9 | [Object.getOwnPropertyNames()](https://www.javatpoint.com/javascript-object-getownpropertynames-method) | This method returns an array of all properties (enumerable or not) found. |
| 10 | [Object.getOwnPropertySymbols()](https://www.javatpoint.com/javascript-object-getownpropertysymbols-method) | This method returns an array of all own symbol key properties. |
| 11 | [Object.getPrototypeOf()](https://www.javatpoint.com/javascript-object-getprototypeof-method) | This method returns the prototype of the specified object. |
| 12 | [Object.is()](https://www.javatpoint.com/javascript-object-is-method) | This method determines whether two values are the same value. |
| 13 | [Object.isExtensible()](https://www.javatpoint.com/javascript-objects) | This method determines if an object is extensible |
| 14 | [Object.isFrozen()](https://www.javatpoint.com/javascript-objects) | This method determines if an object was frozen. |
| 15 | [Object.isSealed()](https://www.javatpoint.com/javascript-objects) | This method determines if an object is sealed. |
| 16 | [Object.keys()](https://www.javatpoint.com/javascript-objects) | This method returns an array of a given object's own property names. |
| 17 | [Object.preventExtensions()](https://www.javatpoint.com/javascript-object-preventextensions-method) | This method is used to prevent any extensions of an object. |
| 18 | [Object.seal()](https://www.javatpoint.com/javascript-object-seal-method) | This method prevents new properties from being added and marks all existing properties as non-configurable. |
| 19 | [Object.setPrototypeOf()](https://www.javatpoint.com/javascript-object-setprototypeof-method) | This method sets the prototype of a specified object to another object. |
| 20 | [Object.values()](https://www.javatpoint.com/javascript-object-values-method) | This method returns an array of values. |

JavaScript Prototype Object

JavaScript is a prototype-based language that facilitates the objects to acquire properties and features from one another. Here, each object contains a prototype object.

In JavaScript, whenever a function is created the prototype property is added to that function automatically. This property is a prototype object that holds a constructor property.

Syntax:

1. ClassName.prototype.methodName

What is the requirement of a prototype object?

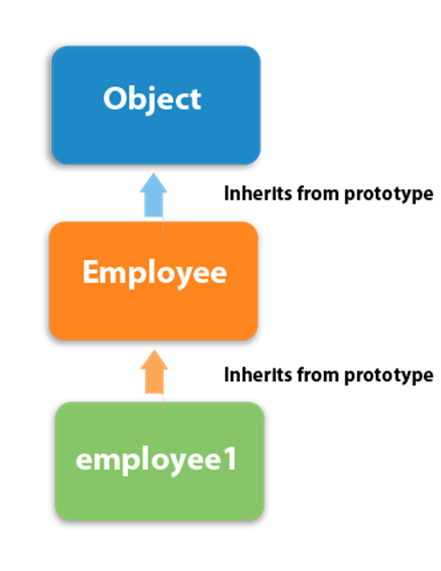
Whenever an object is created in JavaScript, its corresponding functions are loaded into memory. So, a new copy of the function is created on each object creation.

In a prototype-based approach, all the objects share the same function. This ignores the requirement of creating a new copy of function for each object. Thus, the functions are loaded once into the memory.

Play Video

Prototype Chaining

In JavaScript, each object contains a prototype object that acquires properties and methods from it. Again an object's prototype object may contain a prototype object that also acquires properties and methods, and so on. It can be seen as prototype chaining.



JavaScript Prototype Object Example 1

Let's see an example to add a new method to the constructor function.

1. **<script>**
2. function Employee(firstName,lastName)
3. {
4. this.firstName=firstName;
5. this.lastName=lastName;
6. }
8. Employee.prototype.fullName=function()
9. {
10. return this.firstName+" "+this.lastName;
11. }
13. var employee1=new Employee("Martin","Roy");
14. var employee2=new Employee("Duke", "William");
15. document.writeln(employee1.fullName()+"**<br>**");
16. document.writeln(employee2.fullName());
17. **</script>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=JavaScriptPrototypeObjectExample1)

**Output:**

Martin Roy

Duke William

Example 2

Let's see an example to add a new property to the constructor function.

1. **<script>**
2. function Employee(firstName,lastName)
3. {
4. this.firstName=firstName;
5. this.lastName=lastName;
6. }
8. Employee.prototype.company="Javatpoint"
10. var employee1=new Employee("Martin","Roy");
11. var employee2=new Employee("Duke", "William");
12. document.writeln(employee1.firstName+" "+employee1.lastName+" "+employee1.company+"**<br>**");
13. document.writeln(employee2.firstName+" "+employee2.lastName+" "+employee2.company);
14. **</script>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=JavaScriptPrototypeObjectExample2)

**Output:**

Martin Roy Javatpoint

Duke William Javatpoint

# JavaScript Constructor Method

A JavaScript constructor method is a special type of method which is used to initialize and create an object. It is called when memory is allocated for an object.

## Points to remember

* The constructor keyword is used to declare a constructor method.
* The class can contain one constructor method only.
* JavaScript allows us to use parent class constructor through super keyword.

### Constructor Method Example

Let's see a simple example of a constructor method.

1. **<script>**
2. class Employee {
3. constructor() {
4. this.id=101;
5. this.name = "Martin Roy";
6. }
7. }
8. var emp = new Employee();
9. document.writeln(emp.id+" "+emp.name);
10. **</script>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=JavaScriptConstructorMethodExample)

**Output:**

101 Martin Roy

### Constructor Method Example: super keyword

The super keyword is used to call the parent class constructor. Let's see an example.

Play Video

1. **<script>**
2. class CompanyName
3. {
4. constructor()
5. {
6. this.company="Javatpoint";
7. }
8. }
9. class Employee extends CompanyName {
10. constructor(id,name) {
11. super();
12. this.id=id;
13. this.name=name;
14. }
15. }
16. var emp = new Employee(1,"John");
17. document.writeln(emp.id+" "+emp.name+" "+emp.company);
18. **</script>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=JavaScriptConstructorMethodsuperkeywordExample)

**Output:**

1 John Javatpoint

#### **Note - If we didn't specify any constructor method, JavaScript use default constructor method.**

JavaScript static Method

The JavaScript provides static methods that belong to the class instead of an instance of that class. So, an instance is not required to call the static method. These methods are called directly on the class itself.

Points to remember

* The static keyword is used to declare a static method.
* The static method can be of any name.
* A class can contain more than one static method.
* If we declare more than one static method with a similar name, the JavaScript always invokes the last one.
* The static method can be used to create utility functions.
* We can use this keyword to call a static method within another static method.
* We cannot use this keyword directly to call a static method within the non-static method. In such case, we can call the static method either using the class name or as the property of the constructor.

JavaScript static Method Example 1

Let's see a simple example of a static method.

1. **<script>**
2. class Test
3. {
4. static display()
5. {
6. return "static method is invoked"
7. }
8. }
9. document.writeln(Test.display());
10. **</script>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=JavaScriptstaticMethodExample1)

**Output:**

static method is invoked

Example 2

Le's see an example to invoke more than one static method.

Play Video

1. **<script>**
2. class Test
3. {
4. static display1()
5. {
6. return "static method is invoked"
7. }
8. static display2()
9. {
10. return "static method is invoked again"
11. }
12. }
13. document.writeln(Test.display1()+"**<br>**");
14. document.writeln(Test.display2());
15. **</script>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=JavaScriptstaticMethodExample2)

**Output:**

static method is invoked

static method is invoked again

Example 3

Let's see an example to invoke more than one static method with similar names.

1. **<script>**
2. class Test
3. {
4. static display()
5. {
6. return "static method is invoked"
7. }
8. static display()
9. {
10. return "static method is invoked again"
11. }
12. }
13. document.writeln(Test.display());
14. **</script>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=JavaScriptstaticMethodExample3)

**Output:**

static method is invoked again

Example 4

Let's see an example to invoke a static method within the constructor.

1. **<script>**
2. class Test {
3. constructor() {
4. document.writeln(Test.display()+"**<br>**");
5. document.writeln(this.constructor.display());
6. }
8. static display() {
9. return "static method is invoked"
10. }
11. }
12. var t=new Test();
13. **</script>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=JavaScriptstaticMethodExample4)

**Output:**

static method is invoked

static method is invoked

Example 5

Let's see an example to invoke a static method within the non-static method.

1. **<script>**
2. class Test {
3. static display() {
4. return "static method is invoked"
5. }
7. show() {
8. document.writeln(Test.display()+"**<br>**");
9. }
10. }
11. var t=new Test();
12. t.show();
13. **</script>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=JavaScriptstaticMethodExample5)

**Output:**

static method is invoked

JavaScript Encapsulation

The JavaScript Encapsulation is a process of binding the data (i.e. variables) with the functions acting on that data. It allows us to control the data and validate it. To achieve an encapsulation in JavaScript: -

* Use var keyword to make data members private.
* Use setter methods to set the data and getter methods to get that data.

The encapsulation allows us to handle an object using the following properties:

**Read/Write** - Here, we use setter methods to write the data and getter methods read that data.

**Read Only** - In this case, we use getter methods only.

Play Video[](https://campaign.adpushup.com/get-started/?utm_source=banner&utm_campaign=growth_hack)

**Write Only** - In this case, we use setter methods only.

JavaScript Encapsulation Example

Let's see a simple example of encapsulation that contains two data members with its setter and getter methods.

1. **<script>**
2. class Student
3. {
4. constructor()
5. {
6. var name;
7. var marks;
8. }
9. getName()
10. {
11. return this.name;
12. }
13. setName(name)
14. {
15. this.name=name;
16. }
18. getMarks()
19. {
20. return this.marks;
21. }
22. setMarks(marks)
23. {
24. this.marks=marks;
25. }
27. }
28. var stud=new Student();
29. stud.setName("John");
30. stud.setMarks(80);
31. document.writeln(stud.getName()+" "+stud.getMarks());
32. **</script>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=JavaScriptEncapsulationExample)

**Output:**

John 80

JavaScript Encapsulation Example: Validate

In this example, we validate the marks of the student.

1. **<script>**
2. class Student
3. {
4. constructor()
5. {
6. var name;
7. var marks;
8. }
9. getName()
10. {
11. return this.name;
12. }
13. setName(name)
14. {
15. this.name=name;
16. }
18. getMarks()
19. {
20. return this.marks;
21. }
22. setMarks(marks)
23. {
24. if(marks**<0**||marks**>**100)
25. {
26. alert("Invalid Marks");
27. }
28. else
29. {
30. this.marks=marks;
31. }
32. }
33. }
34. var stud=new Student();
35. stud.setName("John");
36. stud.setMarks(110);//alert() invokes
37. document.writeln(stud.getName()+" "+stud.getMarks());
38. **</script>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=JavaScriptEncapsulationValidateExample)

**Output:**

John undefined

JavaScript Encapsulation Example: Prototype-based approach

Here, we perform prototype-based encapsulation.

1. **<script>**
2. function Student(name,marks)
3. {
4. var s\_name=name;
5. var s\_marks=marks;
6. Object.defineProperty(this,"name",{
7. get:function()
8. {
9. return s\_name;
10. },
11. set:function(s\_name)
12. {
13. this.s\_name=s\_name;
14. }
16. });
18. Object.defineProperty(this,"marks",{
19. get:function()
20. {
21. return s\_marks;
22. },
23. set:function(s\_marks)
24. {
25. this.s\_marks=s\_marks;
26. }
28. });
30. }
31. var stud=new Student("John",80);
32. document.writeln(stud.name+" "+stud.marks);
33. **</script>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=JavaScriptEncapsulationPrototypebasedapproachExample)

**Output:**

John 80

JavaScript Inheritance

The JavaScript inheritance is a mechanism that allows us to create new classes on the basis of already existing classes. It provides flexibility to the child class to reuse the methods and variables of a parent class.

The JavaScript **extends** keyword is used to create a child class on the basis of a parent class. It facilitates child class to acquire all the properties and behavior of its parent class.

Points to remember

* It maintains an IS-A relationship.
* The extends keyword is used in class expressions or class declarations.
* Using extends keyword, we can acquire all the properties and behavior of the inbuilt object as well as custom classes.
* We can also use a prototype-based approach to achieve inheritance.

JavaScript extends Example: inbuilt object

In this example, we extends **Date** object to display today's date.

1. **<script>**
2. class Moment extends Date {
3. constructor() {
4. super();
5. }}
6. var m=new Moment();
7. document.writeln("Current date:")
8. document.writeln(m.getDate()+"-"+(m.getMonth()+1)+"-"+m.getFullYear());
9. **</script>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=JavaScriptextendsinbuiltobjectExample1)

**Output:**

Current date: 31-8-2018

Let's see one more example to display the year value from the given date.

1. **<script>**
2. class Moment extends Date {
3. constructor(year) {
4. super(year);
5. }}
6. var m=new Moment("August 15, 1947 20:22:10");
7. document.writeln("Year value:")
8. document.writeln(m.getFullYear());
9. **</script>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=JavaScriptextendsinbuiltobjectExample2)

**Output:**

Year value: 1947

JavaScript extends Example: Custom class

In this example, we declare sub-class that extends the properties of its parent class.

1. **<script>**
2. class Bike
3. {
4. constructor()
5. {
6. this.company="Honda";
7. }
8. }
9. class Vehicle extends Bike {
10. constructor(name,price) {
11. super();
12. this.name=name;
13. this.price=price;
14. }
15. }
16. var v = new Vehicle("Shine","70000");
17. document.writeln(v.company+" "+v.name+" "+v.price);
18. **</script>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=JavaScriptextendsCustomclassExample)

**Output:**

Honda Shine 70000

JavaScript extends Example: a Prototype-based approach

Here, we perform prototype-based inheritance. In this approach, there is no need to use class and extends keywords.

1. **<script>**
2. //Constructor function
3. function Bike(company)
4. {
5. this.company=company;
6. }
8. Bike.prototype.getCompany=function()
9. {
10. return this.company;
11. }
12. //Another constructor function
13. function Vehicle(name,price) {
14. this.name=name;
15. this.price=price;
16. }
17. var bike = new Bike("Honda");
18. Vehicle.prototype=bike; //Now Bike treats as a parent of Vehicle.
19. var vehicle=new Vehicle("Shine",70000);
20. document.writeln(vehicle.getCompany()+" "+vehicle.name+" "+vehicle.price);
21. **</script>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=JavaScriptextendsaPrototypebasedapproachExample1)

**Output:**

Honda Shine 70000

JavaScript Encapsulation

The JavaScript Encapsulation is a process of binding the data (i.e. variables) with the functions acting on that data. It allows us to control the data and validate it. To achieve an encapsulation in JavaScript: -

* Use var keyword to make data members private.
* Use setter methods to set the data and getter methods to get that data.

The encapsulation allows us to handle an object using the following properties:

**Read/Write** - Here, we use setter methods to write the data and getter methods read that data.

**Read Only** - In this case, we use getter methods only.

**Write Only** - In this case, we use setter methods only.

JavaScript Encapsulation Example

Let's see a simple example of encapsulation that contains two data members with its setter and getter methods.

1. **<script>**
2. class Student
3. {
4. constructor()
5. {
6. var name;
7. var marks;
8. }
9. getName()
10. {
11. return this.name;
12. }
13. setName(name)
14. {
15. this.name=name;
16. }
18. getMarks()
19. {
20. return this.marks;
21. }
22. setMarks(marks)
23. {
24. this.marks=marks;
25. }
27. }
28. var stud=new Student();
29. stud.setName("John");
30. stud.setMarks(80);
31. document.writeln(stud.getName()+" "+stud.getMarks());
32. **</script>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=JavaScriptEncapsulationExample)

**Output:**

John 80

JavaScript Encapsulation Example: Validate

In this example, we validate the marks of the student.

1. **<script>**
2. class Student
3. {
4. constructor()
5. {
6. var name;
7. var marks;
8. }
9. getName()
10. {
11. return this.name;
12. }
13. setName(name)
14. {
15. this.name=name;
16. }
18. getMarks()
19. {
20. return this.marks;
21. }
22. setMarks(marks)
23. {
24. if(marks**<0**||marks**>**100)
25. {
26. alert("Invalid Marks");
27. }
28. else
29. {
30. this.marks=marks;
31. }
32. }
33. }
34. var stud=new Student();
35. stud.setName("John");
36. stud.setMarks(110);//alert() invokes
37. document.writeln(stud.getName()+" "+stud.getMarks());
38. **</script>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=JavaScriptEncapsulationValidateExample)

**Output:**

John undefined

JavaScript Encapsulation Example: Prototype-based approach

Here, we perform prototype-based encapsulation.

1. **<script>**
2. function Student(name,marks)
3. {
4. var s\_name=name;
5. var s\_marks=marks;
6. Object.defineProperty(this,"name",{
7. get:function()
8. {
9. return s\_name;
10. },
11. set:function(s\_name)
12. {
13. this.s\_name=s\_name;
14. }
16. });
18. Object.defineProperty(this,"marks",{
19. get:function()
20. {
21. return s\_marks;
22. },
23. set:function(s\_marks)
24. {
25. this.s\_marks=s\_marks;
26. }
28. });
30. }
31. var stud=new Student("John",80);
32. document.writeln(stud.name+" "+stud.marks);
33. **</script>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=JavaScriptEncapsulationPrototypebasedapproachExample)

**Output:**

John 80

JavaScript Inheritance

The JavaScript inheritance is a mechanism that allows us to create new classes on the basis of already existing classes. It provides flexibility to the child class to reuse the methods and variables of a parent class.

The JavaScript **extends** keyword is used to create a child class on the basis of a parent class. It facilitates child class to acquire all the properties and behavior of its parent class.

Points to remember

* It maintains an IS-A relationship.
* The extends keyword is used in class expressions or class declarations.
* Using extends keyword, we can acquire all the properties and behavior of the inbuilt object as well as custom classes.
* We can also use a prototype-based approach to achieve inheritance.

JavaScript extends Example: inbuilt object

In this example, we extends **Date** object to display today's date.

1. **<script>**
2. class Moment extends Date {
3. constructor() {
4. super();
5. }}
6. var m=new Moment();
7. document.writeln("Current date:")
8. document.writeln(m.getDate()+"-"+(m.getMonth()+1)+"-"+m.getFullYear());
9. **</script>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=JavaScriptextendsinbuiltobjectExample1)

**Output:**

Play Video[](https://campaign.adpushup.com/get-started/?utm_source=banner&utm_campaign=growth_hack)

Current date: 31-8-2018

Let's see one more example to display the year value from the given date.

1. **<script>**
2. class Moment extends Date {
3. constructor(year) {
4. super(year);
5. }}
6. var m=new Moment("August 15, 1947 20:22:10");
7. document.writeln("Year value:")
8. document.writeln(m.getFullYear());
9. **</script>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=JavaScriptextendsinbuiltobjectExample2)

**Output:**

Year value: 1947

JavaScript extends Example: Custom class

In this example, we declare sub-class that extends the properties of its parent class.

1. **<script>**
2. class Bike
3. {
4. constructor()
5. {
6. this.company="Honda";
7. }
8. }
9. class Vehicle extends Bike {
10. constructor(name,price) {
11. super();
12. this.name=name;
13. this.price=price;
14. }
15. }
16. var v = new Vehicle("Shine","70000");
17. document.writeln(v.company+" "+v.name+" "+v.price);
18. **</script>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=JavaScriptextendsCustomclassExample)

**Output:**

Honda Shine 70000

JavaScript extends Example: a Prototype-based approach

Here, we perform prototype-based inheritance. In this approach, there is no need to use class and extends keywords.

1. **<script>**
2. //Constructor function
3. function Bike(company)
4. {
5. this.company=company;
6. }
8. Bike.prototype.getCompany=function()
9. {
10. return this.company;
11. }
12. //Another constructor function
13. function Vehicle(name,price) {
14. this.name=name;
15. this.price=price;
16. }
17. var bike = new Bike("Honda");
18. Vehicle.prototype=bike; //Now Bike treats as a parent of Vehicle.
19. var vehicle=new Vehicle("Shine",70000);
20. document.writeln(vehicle.getCompany()+" "+vehicle.name+" "+vehicle.price);
21. **</script>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=JavaScriptextendsaPrototypebasedapproachExample1)

**Output:**

Honda Shine 70000

JavaScript Encapsulation

The JavaScript Encapsulation is a process of binding the data (i.e. variables) with the functions acting on that data. It allows us to control the data and validate it. To achieve an encapsulation in JavaScript: -

* Use var keyword to make data members private.
* Use setter methods to set the data and getter methods to get that data.

The encapsulation allows us to handle an object using the following properties:

**Read/Write** - Here, we use setter methods to write the data and getter methods read that data.

**Read Only** - In this case, we use getter methods only.

**Write Only** - In this case, we use setter methods only.

JavaScript Encapsulation Example

Let's see a simple example of encapsulation that contains two data members with its setter and getter methods.

1. **<script>**
2. class Student
3. {
4. constructor()
5. {
6. var name;
7. var marks;
8. }
9. getName()
10. {
11. return this.name;
12. }
13. setName(name)
14. {
15. this.name=name;
16. }
18. getMarks()
19. {
20. return this.marks;
21. }
22. setMarks(marks)
23. {
24. this.marks=marks;
25. }
27. }
28. var stud=new Student();
29. stud.setName("John");
30. stud.setMarks(80);
31. document.writeln(stud.getName()+" "+stud.getMarks());
32. **</script>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=JavaScriptEncapsulationExample)

**Output:**

John 80

JavaScript Encapsulation Example: Validate

In this example, we validate the marks of the student.

1. **<script>**
2. class Student
3. {
4. constructor()
5. {
6. var name;
7. var marks;
8. }
9. getName()
10. {
11. return this.name;
12. }
13. setName(name)
14. {
15. this.name=name;
16. }
18. getMarks()
19. {
20. return this.marks;
21. }
22. setMarks(marks)
23. {
24. if(marks**<0**||marks**>**100)
25. {
26. alert("Invalid Marks");
27. }
28. else
29. {
30. this.marks=marks;
31. }
32. }
33. }
34. var stud=new Student();
35. stud.setName("John");
36. stud.setMarks(110);//alert() invokes
37. document.writeln(stud.getName()+" "+stud.getMarks());
38. **</script>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=JavaScriptEncapsulationValidateExample)

**Output:**

John undefined

JavaScript Encapsulation Example: Prototype-based approach

Here, we perform prototype-based encapsulation.

1. **<script>**
2. function Student(name,marks)
3. {
4. var s\_name=name;
5. var s\_marks=marks;
6. Object.defineProperty(this,"name",{
7. get:function()
8. {
9. return s\_name;
10. },
11. set:function(s\_name)
12. {
13. this.s\_name=s\_name;
14. }
16. });
18. Object.defineProperty(this,"marks",{
19. get:function()
20. {
21. return s\_marks;
22. },
23. set:function(s\_marks)
24. {
25. this.s\_marks=s\_marks;
26. }
28. });
30. }
31. var stud=new Student("John",80);
32. document.writeln(stud.name+" "+stud.marks);
33. **</script>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=JavaScriptEncapsulationPrototypebasedapproachExample)

**Output:**

John 80

JavaScript Polymorphism

The polymorphism is a core concept of an object-oriented paradigm that provides a way to perform a single action in different forms. It provides an ability to call the same method on different JavaScript objects. As JavaScript is not a type-safe language, we can pass any type of data members with the methods.

JavaScript Polymorphism Example 1

Let's see an example where a child class object invokes the parent class method.

1. **<script>**
2. class A
3. {
4. display()
5. {
6. document.writeln("A is invoked");
7. }
8. }
9. class B extends A
10. {
11. }
12. var b=new B();
13. b.display();
14. **</script>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=JavaScriptPolymorphismExample1)

**Output:**

A is invoked

Example 2

Let's see an example where a child and parent class contains the same method. Here, the object of child class invokes both classes method.

1. **<script>**
2. class A
3. {
4. display()
5. {
6. document.writeln("A is invoked**<br>**");
7. }
8. }
9. class B extends A
10. {
11. display()
12. {
13. document.writeln("B is invoked");
14. }
15. }
17. var a=[new A(), new B()]
18. a.forEach(function(msg)
19. {
20. msg.display();
21. });
22. **</script>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=JavaScriptPolymorphismExample2)

**Output:**

A is invoked

B is invoked

Example 3

Let's see the same example with prototype-based approach.

1. **<script>**
2. function A()
3. {
4. }
5. A.prototype.display=function()
6. {
7. return "A is invoked";
8. }
9. function B()
10. {
12. }
14. B.prototype=Object.create(A.prototype);
16. var a=[new A(), new B()]
18. a.forEach(function(msg)
19. {
20. document.writeln(msg.display()+"**<br>**");
21. });
22. **<script>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=JavaScriptPolymorphismExample3)

**Output:**

A is invoked

B is invoked

JavaScript Abstraction

An abstraction is a way of hiding the implementation details and showing only the functionality to the users. In other words, it ignores the irrelevant details and shows only the required one.

Points to remember

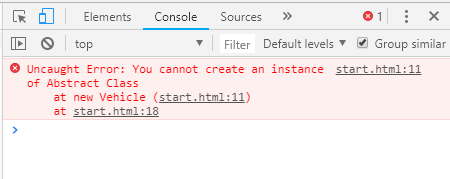
* We cannot create an instance of Abstract Class.
* It reduces the duplication of code.

JavaScript Abstraction Example

Example 1

Let's check whether we can create an instance of Abstract class or not.

1. **<script>**
2. //Creating a constructor function
3. function Vehicle()
4. {
5. this.vehicleName= vehicleName;
6. throw new Error("You cannot create an instance of Abstract class");
8. }
9. Vehicle.prototype.display=function()
10. {
11. return this.vehicleName;
12. }
13. var vehicle=new Vehicle();
14. **</script>**



Example 2

Let's see an example to achieve abstraction.

1. **<script>**
2. //Creating a constructor function
3. function Vehicle()
4. {
5. this.vehicleName="vehicleName";
6. throw new Error("You cannot create an instance of Abstract Class");
7. }
8. Vehicle.prototype.display=function()
9. {
10. return "Vehicle is: "+this.vehicleName;
11. }
12. //Creating a constructor function
13. function Bike(vehicleName)
14. {
15. this.vehicleName=vehicleName;
16. }
17. //Creating object without using the function constructor
18. Bike.prototype=Object.create(Vehicle.prototype);
19. var bike=new Bike("Honda");
20. document.writeln(bike.display());

23. **</script>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=JavaScriptAbstractionExample2)

**Output:**

Vehicle is: Honda

Example 3

In this example, we use instanceof operator to test whether the object refers to the corresponding class.

1. **<script>**
2. //Creating a constructor function
3. function Vehicle()
4. {
5. this.vehicleName=vehicleName;
6. throw new Error("You cannot create an instance of Abstract class");
7. }
8. //Creating a constructor function
9. function Bike(vehicleName)
10. {
11. this.vehicleName=vehicleName;
12. }
13. Bike.prototype=Object.create(Vehicle.prototype);
14. var bike=new Bike("Honda");
15. document.writeln(bike instanceof Vehicle);
16. document.writeln(bike instanceof Bike);
18. **</script>**

[**Test it Now**](https://www.javatpoint.com/oprweb/test.jsp?filename=JavaScriptAbstractionExample3)

**Output:**

true true