**Objects and its internal representation in Javascript**

**Introduction:**

JavaScript, with its versatility and dynamic nature, heavily relies on objects for data representation and manipulation. Understanding how objects are internally represented in JavaScript is fundamental for every developer aiming to write efficient and maintainable code. In this blog post, we'll delve into the intricacies of objects, their internal structure, and how JavaScript engines handle them under the hood.

**Objects in JavaScript:**

In JavaScript, an object is a collection of key-value pairs, where each key is a string (or Symbol) and each value can be of any data type, including other objects or functions. Objects in JavaScript can be created using object literals, constructors, or classes.

**Sample code:**

// Object literal

const person = {

name: "Bala",

age: 32,

greet() {

console.log(`Hello, my name is ${this.name}`);

}

};

// Constructor

function Car(make, model) {

this.make = make;

this.model = model;

}

// Class

class Animal {

constructor(name) {

this.name = name;

}

}

**Internal Representation of Objects:**

**Properties and Prototype Chain:**

In JavaScript, objects have properties, which are essentially key-value pairs. Each object also has a prototype, which is another object that serves as a fallback source for property lookup. When a property is accessed on an object, the JavaScript engine first looks for that property directly on the object. If it's not found, it looks up the prototype chain until the property is found or until the end of the chain is reached.

**Property Descriptors:**

Every property of an object in JavaScript is associated with a property descriptor, which defines attributes such as value, writability, configurability, and enumerability. These attributes determine how the property behaves and can be accessed or modified using methods like Object.getOwnPropertyDescriptor() and Object.defineProperty().

**Hidden Classes and Shapes:**

JavaScript engines optimize object property access and memory usage using hidden classes (V8 engine) or shapes (SpiderMonkey engine). These internal representations help improve performance by efficiently storing and accessing object properties. When objects share the same structure (i.e., same properties in the same order), they can be optimized to use the same hidden class/shape, reducing memory overhead and speeding up property access.