JavaScript Classes Review

asics of Working with Classes

- **Definition**: Classes in JavaScript are used to define blueprints for creating objects, and encapsulating data. Classes include a constructor which is a special method that gets called automatically when a new object is created from the class. It is used to initialize the properties of the object. The this keyword is used here to refer to the current instance of the class. Below the constructor, you can have what are called methods. Methods are functions defined inside a class that perform actions or operations on the class's data or state. They are used to define behaviors that instances of the class can perform.

Example Code

```
class Dog {
 constructor(name) {
  this.name = name;
 }

 bark() {
  console.log(`${this.name} says woof!`);
 }
}
```

To create a new instance of the class, you will use the new keyword followed by the class name:

Example Code

```
const dog = new Dog("Gino");
```

You can also create classes as class expressions. This is where the class is anonymous and assigned to a variable.

Example Code

```
const Dog = class {
 constructor(name) {
  this.name = name;
 }

 bark() {
  console.log(`${this.name} says woof!`);
 }
};
```

Class Inheritance

- **Definition**: In programming, inheritance allows you to define classes that inherit properties and methods from parent classes. This promotes code reuse and establishes a hierarchical relationship between classes. A parent class is a class that acts like a blueprint for other classes. It defines properties and methods that are inherited by other classes. A child class is a class that inherits the properties and methods of another class. Child classes can also extend the functionality of their parent classes by adding new properties and methods. In JavaScript, we use the extends keyword to implement inheritance. This keyword indicates that a class is the child class of another class.

Example Code

```
class Vehicle {
 constructor(brand, year) {
  this.brand = brand;
  this.year = year;
 }
}

class Car extends Vehicle {
 honk() {
    console.log("Honk! Honk!");
  }
}
```

The super keyword is used to access the parent class's methods, constructors, and fields.

Example Code

```
class Vehicle {
  constructor(brand, year) {
    this.brand = brand;
    this.year = year;
  }
}

class Car extends Vehicle {
  constructor(brand, year, numDoors) {
    super(brand, year);
    this.numDoors = numDoors;
  }
}
```

Working with Static Methods and Static Properties

- **Static methods**: These methods are often used for utility functions that don't need access to the specific state of an object. They are defined within classes to encapsulate related functionality. Static methods are also helpful for implementing "factory" methods. A factory method is a method that you define in addition to the constructor to create objects based on specific criteria.

Example Code

```
class Movie {
  constructor(title, rating) {
```

```
  this.title = title;
  this.rating = rating;
 }

 static compareMovies(movieA, movieB) {
  if (movieA.rating > movieB.rating) {
   console.log(` ${movieA.title} has a higher rating.` );
  } else if (movieA.rating < movieB.rating) {
   console.log(` ${movieB.title} has a higher rating.` );
  } else {
   console.log("These movies have the same rating.");
  }
 }
}

let movieA = new Movie("Movie A", 80);
let movieB = new Movie("Movie B", 45);

Movie.compareMovies(movieA, movieB);
```

- **Static Properties**: These properties are used to define values or attributes that are associated with a class itself, rather than with instances of the class. Static properties are shared across all instances of the class and can be accessed without creating an instance of the class.

Example Code

```
class Car {
 // Static property
 static numberOfWheels = 4;
 constructor(make, model) {
  this.make = make;
  this.model = model;
 }

 // Instance method
 getCarInfo() {
  return `${this.make} ${this.model}`;
 }

 // Static method
 static getNumberOfWheels() {
  return Car.numberOfWheels;
 }
}

// Accessing static property directly from the class
console.log(Car.numberOfWheels);
```