Core JavaScript Mastery Guide

Perfect 👌 You've shared the **JavaScript Classes** notes, so let's now turn them into a **structured, in-depth guide** with all **important points, explanations, interview insights, and examples**.

📘 JavaScript Classes (In Depth)

1. What is a Class in JavaScript?

- A **class** is a **blueprint/template** for creating objects.
- Introduced in **ES6 (2015)**.
- Classes are **syntactic sugar** over JavaScript's prototype-based inheritance.

👉 Internally, classes are still **functions** with a `prototype`.

```
js

class Car {}
console.log(typeof Car); // "function"
```

---

2. Why Classes?

Before ES6, object blueprints were created using **constructor functions**:

```
js

function Car(brand) {
  this.brand = brand;
}
const car1 = new Car("Audi");
```

With ES6 classes, the same is written more cleanly:

```
js

class Car {
  constructor(brand) {
    this.brand = brand;
  }
```

```
}
const car1 = new Car("Audi");
```

✅ Easier to read, maintain, and extend.

---

3. Class Declaration vs Class Expression

🔹 Class Declaration

```
js

class Car {
  constructor(brand) {
    this.brand = brand;
  }
}
```

🔹 Class Expression

```
js

const Car = class {
  constructor(brand) {
    this.brand = brand;
  }
};
```

⚠ Difference: **Declarations are not hoisted**, so you must define them before use.

---

4. The `constructor()` Method

- Special method in a class.
- Runs automatically when a new object is created with `new`.
- Used for **property initialization**.

```
js

class Car {
  constructor(brand, model) {
    this.brand = brand;
```

```
    this.model = model;
  }
}

const car1 = new Car("BMW", "X5");
console.log(car1.brand); // BMW
```

⚠ Each class can have only **one** `constructor()`.

---

5. Creating Objects from Classes

```
js

class Car {
  constructor(brand, model, year = 2020) {
    this.brand = brand;
    this.model = model;
    this.year = year;
  }
}

const car1 = new Car("Audi", "A6", 2022);
const car2 = new Car("BMW", "X5"); // year defaults to 2020
```

---

6. Class Methods

- Functions inside classes.
- No `function` keyword needed.

```
js

class Car {
  constructor(brand, price) {
    this.brand = brand;
    this.price = price;
  }
  updatePrice(newPrice) {
    this.price = newPrice;
  }
}
```

```
const car = new Car("Tesla", 50000);
car.updatePrice(55000);
console.log(car.price); // 55000
```

---

7. Static Methods

- Belong to the **class itself**, not instances.
- Useful for utility/helper functions.

```
js

class MathUtil {
  static add(a, b) {
    return a + b;
  }
}

console.log(MathUtil.add(5, 10)); // 15
```

⚠ You cannot call `add()` on an instance → only on the class.

---

8. Inheritance (extends, super)

- One class can inherit from another.

```
js

class Vehicle {
  constructor(type) {
    this.type = type;
  }
  start() {
    console.log(`${this.type} is starting`);
  }
}

class Car extends Vehicle {
  constructor(brand, model) {
    super("Car"); // call parent constructor
    this.brand = brand;
```

```
    this.model = model;
  }
}

const car1 = new Car("BMW", "X7");
car1.start(); // Car is starting
```

---

9. Getters and Setters

Encapsulation: control how properties are accessed/updated.

```
js

class User {
  constructor(name, age) {
    this._name = name;
    this._age = age;
  }

  get age() {
    return this._age;
  }

  set age(newAge) {
    if (newAge < 0) throw new Error("Age cannot be negative");
    this._age = newAge;
  }
}

const user1 = new User("Anoop", 21);
console.log(user1.age); // 21
user1.age = 25;
```

---

10. Private Fields (`#`)

- Introduced in **ES2022**.
- Makes properties truly private.

```
js
```

```
class BankAccount {
  #balance = 0; // private field

  deposit(amount) {
    this.#balance += amount;
  }

  getBalance() {
    return this.#balance;
  }
}

const acc = new BankAccount();
acc.deposit(100);
console.log(acc.getBalance()); // 100
// console.log(acc.#balance); ❌ Error
```

---

11. Important Rules

1. Classes are **not hoisted**.
2. Always run in **strict mode**.
3. Can only have **one constructor**.
4. Can use **extends + super** for inheritance.
5. Support **static, getters, setters, private fields**.

---

12. Interview Questions

1. **Are classes in JavaScript really classes?**
   → No, they are **functions** with prototype-based inheritance.
2. **Difference between `class` and `constructor function`?**
   - Class is **syntactic sugar** with cleaner syntax.
   - Constructor function can be called without `new`, classes cannot.
3. **Can we use multiple constructors in JS classes?**
   → No. Only one `constructor()` is allowed. Use **default params** or **overloaded logic**.
4. **Difference between `static` and instance methods?**
   - Static → belongs to class itself.

- Instance → belongs to object created from class.

5. **Why classes run in strict mode?**
   → To avoid accidental globals and silent errors.

✅ With this, you have a **complete foundation + advanced topics of Classes**.

Do you want me to move next into **Object-Oriented Concepts with Classes** (like **Encapsulation, Abstraction, Polymorphism in JS**) with **examples + interview questions**?

2/2