Class constructor and static

Video: Class Constructor and Static in JavaScript – Chai aur Code

📦 `class` and `constructor` in JavaScript

```
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }
}
```

🔷 `constructor()`:

- A special method inside a class that runs **automatically** when you create a new object using `new`.
- It's used to initialize the object's properties.
- **Only one** constructor is allowed per class.
- If no constructor is defined, JavaScript provides a default empty one.

❗ Important:

- You **cannot** make a constructor `async`.
- In inherited classes, always call `super()` first before using `this`.

🧬 Inheritance and `super()`

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

```
class Dog extends Animal {
  constructor(name, breed) {
    super(name);        // Calls the constructor of Animal
    this.breed = breed;
  }
}
```

🔷 `super()`:

- Used to call the parent class's constructor in a child class.
- **Must be called before using** `this` in the child constructor.
- Helps in reusing and extending the parent class logic.

⚙️ `static` Methods and Properties

```
class MathUtils {
  static PI = 3.14;

  static square(x) {
    return x * x;
  }

  static circleArea(radius) {
    return MathUtils.PI * radius * radius;
  }
}
```

🔷 What is `static`?

- The `static` keyword is used to define **class-level** methods or properties.
- These are **not accessible on objects** created from the class.
- They are called **on the class itself**, like `MathUtils.square(4)`.

🔷 Key Rules:

- `static` methods do **not have access to** `this` **from instance** scope.
- They are typically used for:
  - **Utility/helper methods** (e.g., `Math.square()`)
  - **Constants** or shared configuration (e.g., `Config.API_KEY`)
- You can access static members **only using the class name**, not via an instance.

```
console.log(MathUtils.square(5)); // ✅ 25
const util = new MathUtils();
util.square(5); // ❌ Error
```

✅ Example: Full Class Using Constructor, Inheritance, and Static

```
class Vehicle {
  constructor(type) {
    this.type = type;
  }

  describe() {
    return `This is a ${this.type}`;
  }

  static company() {
    return "AutoCorp Ltd.";
  }
}

class Car extends Vehicle {
  constructor(type, model) {
    super(type);
    this.model = model;
  }
```

```
  getDetails() {
    return `${this.describe()} of model ${this.model}`;
  }
}

const myCar = new Car("Sedan", "Model X");
console.log(myCar.getDetails());     // This is a Sedan of model Model X
console.log(Car.company());         // AutoCorp Ltd.
```

⚠️ Common Errors and Gotchas

Mistake	What Happens
Not calling `super()` in child constructor	❌ Throws an error
Defining multiple constructors	❌ Not allowed in JS
Calling static method on instance	❌ TypeError
Using `this` in static method	❌ Will not refer to object instance

📝 Quick Revision Notes (Copy-Paste Friendly)

```
🧱 class MyClass {
  constructor(a, b) {
    this.a = a;
    this.b = b;
  }

  instanceMethod() {
    console.log(`A is ${this.a}`);
  }

  static staticMethod(x) {
    return x * 2;
  }
```

```
  static staticValue = 100;
}
```

🟡 constructor():
- Runs automatically on `new ClassName()`
- Used to initialize object state
- Only one constructor allowed

🟡 super():
- Required in subclass constructor before `this`
- Calls parent class constructor logic

🟢 static:
- Defined using `static` keyword
- Belongs to the class, not the instances
- Called like: `MyClass.staticMethod()`
- Can't use instance properties or `this` (unless also static)
- Good for:
  - Shared constants
  - Utility/helper functions
- Inherited by subclasses

📌 Example:

```
class Config {
  static API_KEY = "xyz-1234";
}
console.log(Config.API_KEY); // ✅ xyz-1234

class Utility {
  static greet() {
    console.log("Hello!");
  }
}
```

```
Utility.greet(); // ✅
new Utility().greet(); // ❌ Error
```