Day 4 — this, Prototypes & Object Model

1. this Binding Rules

a) Default Binding
```
function show() { console.log(this); }
show(); // window or undefined (strict mode)
```

b) Implicit Binding
```
const user = { name: "Ali", hi() { console.log(this.name); } };
user.hi(); // Ali
```

c) Explicit Binding (call/apply/bind)
```
function greet() { console.log(this.name); }
greet.call({name:"Sara"}); // Sara
```

d) new Binding
```
function Person(n){ this.name=n; }
new Person("Omar"); // this = new object
```

e) Arrow Functions
```
const obj={name:"Mona", f:()=>console.log(this)};
obj.f(); // not obj
```

2. call / apply / bind

call: `fn.call(obj, a, b)`
apply: `fn.apply(obj, [a, b])`
bind: `const f = fn.bind(obj)`

Example:
```
function hi(){ console.log(this.name); }
hi.call({name:"Nada"}); // Nada
```

3. Method Borrowing

```
const person1={name:"Ali", hi(){console.log(this.name);} };
const person2={name:"Omar"};
person1.hi.call(person2); // Omar
```

4. Prototype Chain

```
const obj = {}; // obj → Object.prototype → null
```

Searching for properties goes upward the chain.

5. Constructor Pattern

```
function Car(b, m){ this.brand=b; this.model=m; }
Car.prototype.info = function(){ return this.brand + " " + this.model; };

const c1 = new Car("Toyota","Corolla");
c1.info(); // Toyota Corolla
```

6. Summary

- this depends on how a function is called.
- call/apply/bind allow manual binding.
- new creates a fresh object and binds this to it.
- Prototypes allow shared methods across instances.