CALL
The call method in JavaScript allows you to invoke a function on an object while providing a specific context (this value) for that function to use, along with optional arguments that the function can accept. This enables dynamic context binding and the passing of specific values directly to the function being called

APPLY
Apply and Call method are almost same. The only difference is, when we are using apply method, we need to pass an array as the argument.

BIND
The bind method in JavaScript creates a new function that, when invoked, has a specific this value set to a provided value. It's useful for "binding" a function to a particular object as its context. The new function produced by bind can also have preset arguments if provided.

SHALLOW COPY
A shallow copy creates a new object or array, but the contents within it remain references to the original data. It copies the top-level structure without recursively copying nested objects or arrays.

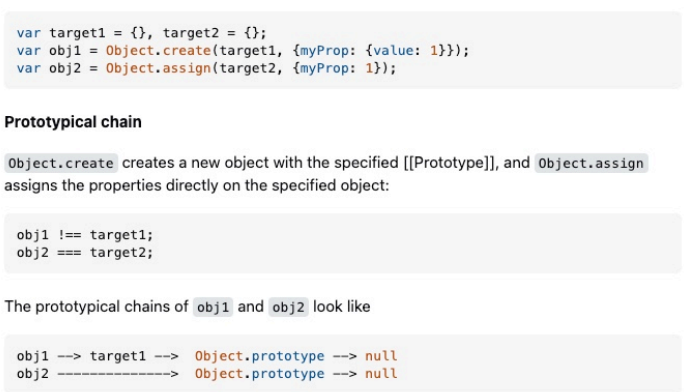
DEEP COPY
A deep copy creates a completely independent copy of an object or array, including all nested objects or arrays. It ensures that changes made to the copied structure do not affect the original.

ABSTRACTION
Abstraction means hiding certain details that don't matter to the user and only showing essential features or functions.

ENCAPSULATION
Encapsulation means keeping properties and methods private inside a class, so that they are not accessible from outside that class.

POLYMORPHISM
Polymorphism means having different and many forms. We can overwrite a method inherited from a parent class.

OBJECT.CREATE VS OBJECT.ASSIGN



```
var target1 = {}, target2 = {};
var obj1 = Object.create(target1, {myProp: {value: 1}});
var obj2 = Object.assign(target2, {myProp: 1});
```

Prototypical chain

`Object.create` creates a new object with the specified [[Prototype]], and `Object.assign` assigns the properties directly on the specified object:

```
obj1 !== target1;
obj2 === target2;
```

The prototypical chains of `obj1` and `obj2` look like

```
obj1 --> target1 -->  Object.prototype --> null
obj2 ---------------->  Object.prototype --> null
```

Agenda of the class

1. OOPS pillars
   - ↳ polymorphism
   - ↳ encapsulation
   - ↳ abstraction
2. Call, Bind & Apply
3. Memory of Object
   - ↳ primitive & reference data types
   - ↳ shallow copy & deep copy

Memory in JS

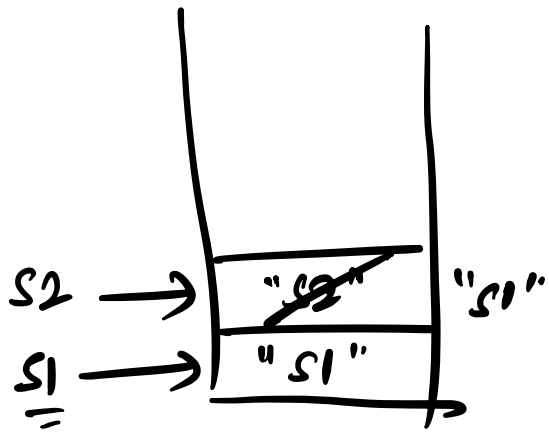
1) Stack memory
2) Heap memory.

```
let s1 = "s1"
let s2 = "s2"
```

s2 = s1

↓

update the value of s2 with the value of s1

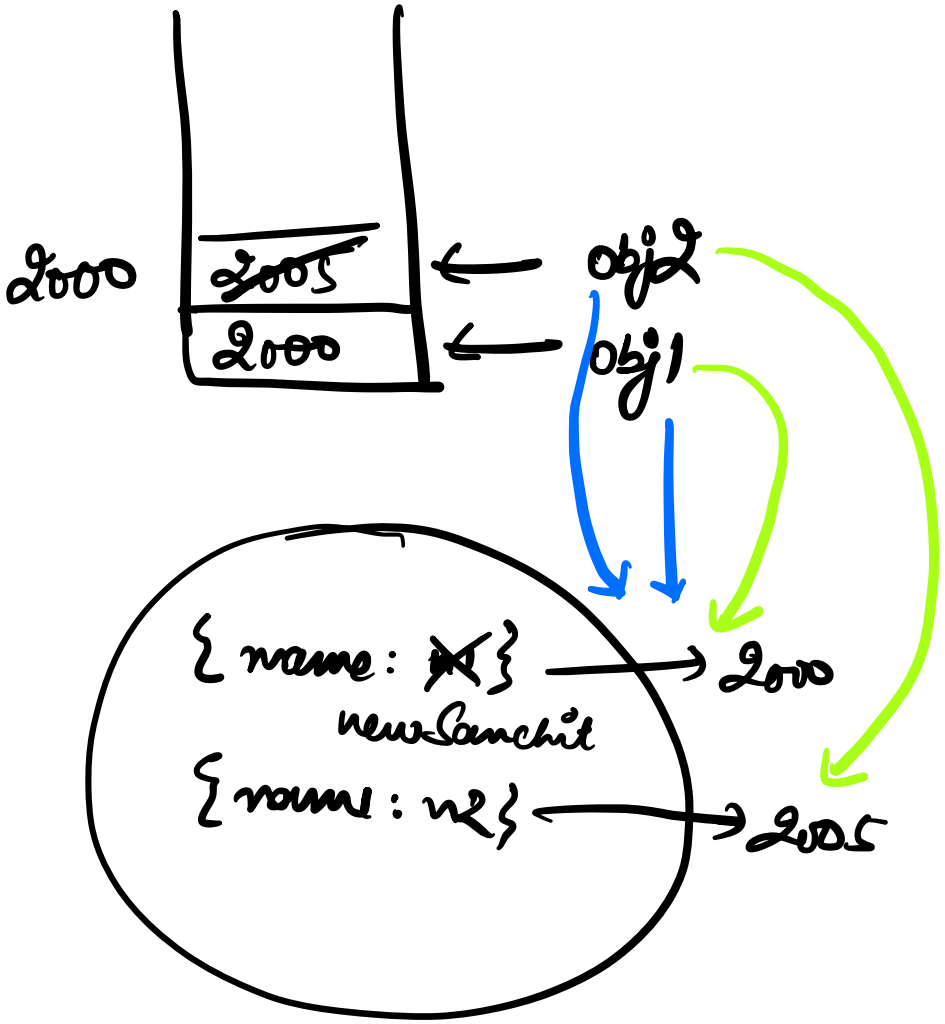


updating by value

```
let obj1 = {
    name: "n1"
}

let obj2 = {
    name: "n2"
}

obj2 = obj1
```




```
Object.create()

obj1 = {

}
```

let obj2 = Object.create(obj1)