Implementar l'herència
mètodes de herència

- **<u>Emmascarar objectes amb el paradigma de constructors:</u>**
- Com totes les propietats i mètodes estan definits amb la paraula clau **this,** es pot convertir el constructor de la classe pare (ClassA) en un mètode de la classe filla (ClassB): **this.newMethod = ClassA**;
- A continuació es pot assignar un paràmetre al mètode creat: **this.newMethod(sColor)**
- I s'elimina la referència a ClassA perquè no es torni a invocar posteriorment.

  delete this.newMethod

```
function ClassA(sColor) {
    this.color = sColor;
    this.sayColor = function () {
        alert(this.color);
    };
}

function ClassB(sColor, sNom) {
    this.newMethod = ClassA;
    this.newMethod(sColor);
    delete this.newMethod;
    this.nom = sNom;
    this.sayName = function () {
        alert(this.nom);
    };
}

var objA = new ClassA("vermell");
var objB = new ClassB("blau", "Marc");
objA.sayColor();
objB.sayColor();
objB.sayName();
```

Implementar l'herència
mètodes de herència

- **<u>Emmascarar objectes amb el paradigma de constructors:</u>**
- El mètode **call()** s'utilitza per emmascarar objectes i accepta dos arguments, el primer és l'objecte utilitzar per a **this**, i el segon es passa directament a la funció.

  En l'exemple veiem que **call()** substitueix les tres línies utilitzades per l'emmascarament en l'exemple anterior.
- El mètode **apply()** és semblant a **call()**, però en el segon argument admet un **Array**:

  ClassA,apply(this, new Array(sColor);

```
function ClassB(sColor, sNom) {
    // this.newMethod = ClassA;
    //this.newMethod(sColor);
    //delete this.newMethod;
    //Mètode call()
    ClassA.call(this, sColor);
    //Mètode apply()
    ClassA.apply(this, new Array(sColor));
    this.nom = sNom;
    this.sayName = function () {
        alert(this.nom);
    };
}

var objA = new ClassA("vermell");
var objB = new ClassB("blau", "Marc");
objA.sayColor();
objB.sayColor();
objB.sayName();
```

Implementar l'herència
mètodes de herència

- **Emmascarar objectes amb el paradigma híbrid:**
- En aquest mètode per crear l'herència s'utilitzen objectes emmascarats per heretar les propietats del constructor, i els prototips per heretar propietats del objecte **prototype**.

```
function ClassA(sColor) {
    this.color = sColor;
}
ClassA.prototype.sayColor = function () {
        alert(this.color);
};
function ClassB(sColor, sNom) {
    // hereta les propietats de ClassA
    ClassA.call(this, sColor);
    this.nom = sNom;
}
// hereta els mètodes de ClassA
ClassB.prototype = new ClassA();
// defineix un nou mètode per ClassB
ClassB.prototype.sayName = function () {
    alert(this.nom);
};

var objA = new ClassA("vermell");
var objB = new ClassB("blau", "Marc");
objA.sayColor();
objB.sayColor();
objB.sayName();
```

Implementar l'herència

Un exemple pràctic

- **Crear la classe base Poligon:**

```
function Poligon(iCostats) {
    this.Costats  = iCostats;
}

Poligon.prototype.getArea = function () {
    return 0;
};
```

- **Crear la subclasse Triangle:**

```
function Triangle(iBase, iAltura) {
    Poligon.call(this, 3);
    this.base = iBase;
    this.altura = iAltura;
}

Triangle.prototype.getArea = function () {
    return 0.5 * this.base * this.altura;
};
```

Implementar l'herència

Un exemple pràctic

- **Crear la subclasse Rectangle:**

```
function Rectangle(iLongitud, iLlargada) {
    Poligon.call(this, 4);
    this.length = iLongitud;
    this.width = iLlargada;
}

Rectangle.prototype.getArea = function () {
    return this.length * this.width;
};
```

- **Comprovar el funcionament:**

```
var triangle = new Triangle(12, 4);
var rectangle = new Rectangle(22, 10);

alert(triangle.Costats);
alert(triangle.getArea());

alert(rectangle.Costats);
alert(rectangle.getArea());
```