Расширенная работа с объектами. ООП в Javascript

Встроенные объекты JS

- String
- Array
- Math
- Date
- Number
- Boolean
- Function
- Object
- JSON

Number

- Number.**MAX_VALUE** - максимально возможное число
- Number.**MIN_VALUE** - минимально возможное число
- Number.**parseInt**( ) - преобразование строки в число
- Number.**parseFloat**( ) - преобразование строки в число с запятой
- Number.**isNaN**( ) - метод для проверки значения на NaN

Math

JavaScript-объект Math используется для выполнения математических функций. Создать экземпляр этого объекта нельзя. Некоторые свойства:

- **Math.PI**
- **Math.pow** - возведение числа в степень
- **Math.abs** - возращает число по модулю
- .....

JSON

Объект которы предоставляет методы для конвертации объектов в строковое представлений и для обратной конвертации из строк объектов.

- JSON.parse - создание обекта из строки
- JSON.stringify - конвертация объекта в строку

Date

JavaScript-объект Date позволяет работать с датами и временем. Этот объект можно создать разными путями.

var myDate = new Date(); - простое создание

- getDate
- getDay
- getFullYear
- setHours
- getMilliseconds
- getMinutes
- setMonth
- getSeconds
- getTime
- getTimezoneOffset
- .....

Object

Представляет собой базовый объект с набором методов которые можно применить к любому объекту. Набор методов:

- `Object.hasOwnProperty` - проверяет наличие свойства в объекте
- Object.keys - возращает список свойств объекта
- Object. defineProperty - расширенное добавления свойства в объект
- `Object.freeze` - замораживает объект, невозможно добавлять свойства и менять их
- Object.seal - запрещает добавления новых свойств.
- ....

ООП. Наследование

Механизм языка, позволяющий описать новый класс на основе уже существующего (родительского, базового) класса. Класс-потомок может добавить собственные методы и свойства, а также пользоваться родительскими методами и свойствами. Позволяет строить иерархии классов. По умолчанию в JS не встроен класический механизм наследование.

Вариант реализации наследования в JS:

- На основе копирования свойств
- Заимствование конструктора
- Совместное использование прототипов
- Смешаный
- Через промежуточный конструктор

Прототип (prototype)

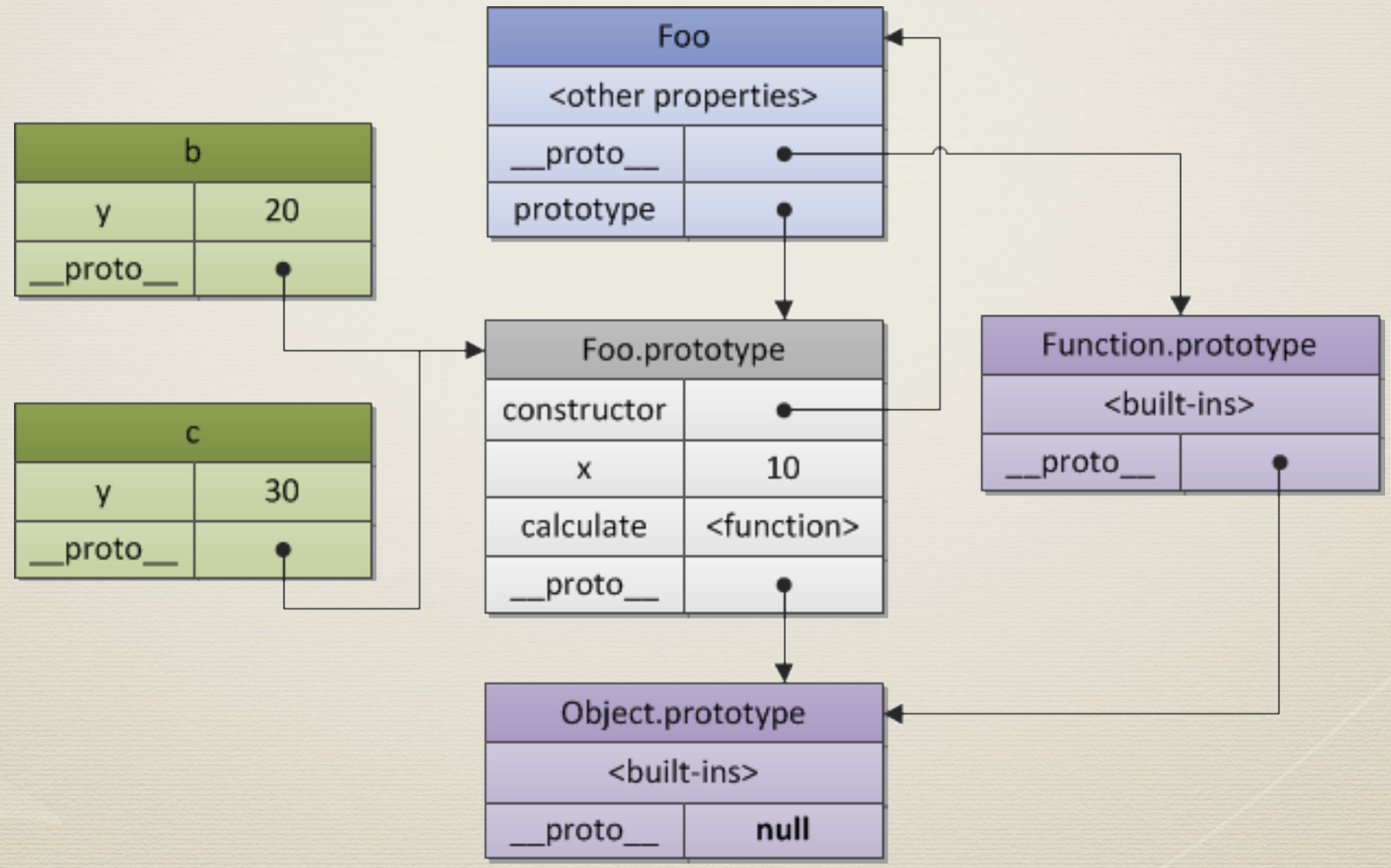
Свойство `prototype` используется для предоставления базового набора функциональных возможностей классу объектов. Новые экземпляры объекта наследуют поведение прототипа, присвоенного этому объекту. Протип является одним для все экземпляров созданных с помощью этого конструктора и изменение в прототипе одного экземпляра будет вижно во всех экземплярах. Свойство prototype есть у каждой функции даже если его не устанавливали. Внутри созданного объекта свойство `__proto__` ссылается на его прототип.

```
function Car() {};
Car.prototype.go = function () { console.log('we are running') };
var car1 = new Car();
var car2 = new Car();

car1.go();
car2.go();
```

Цепочка прототипов

При разрешении имен свойств внутри методов действует цепочка прототипов. Это означает что сначала осуществляется поиск свойства с заданным именем сначала внутри объекта потом внутри прототипа, потом внутри родителя прототипа и так далее вплоть по цепочка до объекта Object



Расширение встроенных прототипов

В JS есть возможность расширять прототипы встроенных объектов таких как Array, Object, String и такой функционал станет доступен во всех экземплярах этих объектов. Такой прием очень нежелателен и используется для кроссбраузерности или для написания функционала который появится в следующих версиях JS.

```
if (typeof Object.prototype.myMethod !== “function”) {
        Object.prototype.myMethod = function () {
            // реализация...
        };
}


var cc = {};
cc.myMethod();
```

Constructor

Свойство constructor является ссылкой на функцию которая создает объект. Это единственное свойство которое есть по умолчанию в prototype новой функции которая еще не была унаследована. Внутри созданного объекта получить ссылку на `constructor` можно через свойство contsructor объекта. При наследовании через прототипы может быть переписан.

```
function Class();
var classInstance = new Class();
console.log(classInstance.contructor == Class)
```

Наследование через копирование свойств

Самый простой способ наследование когда все свойства родительского объекта копируются в дочерний объект

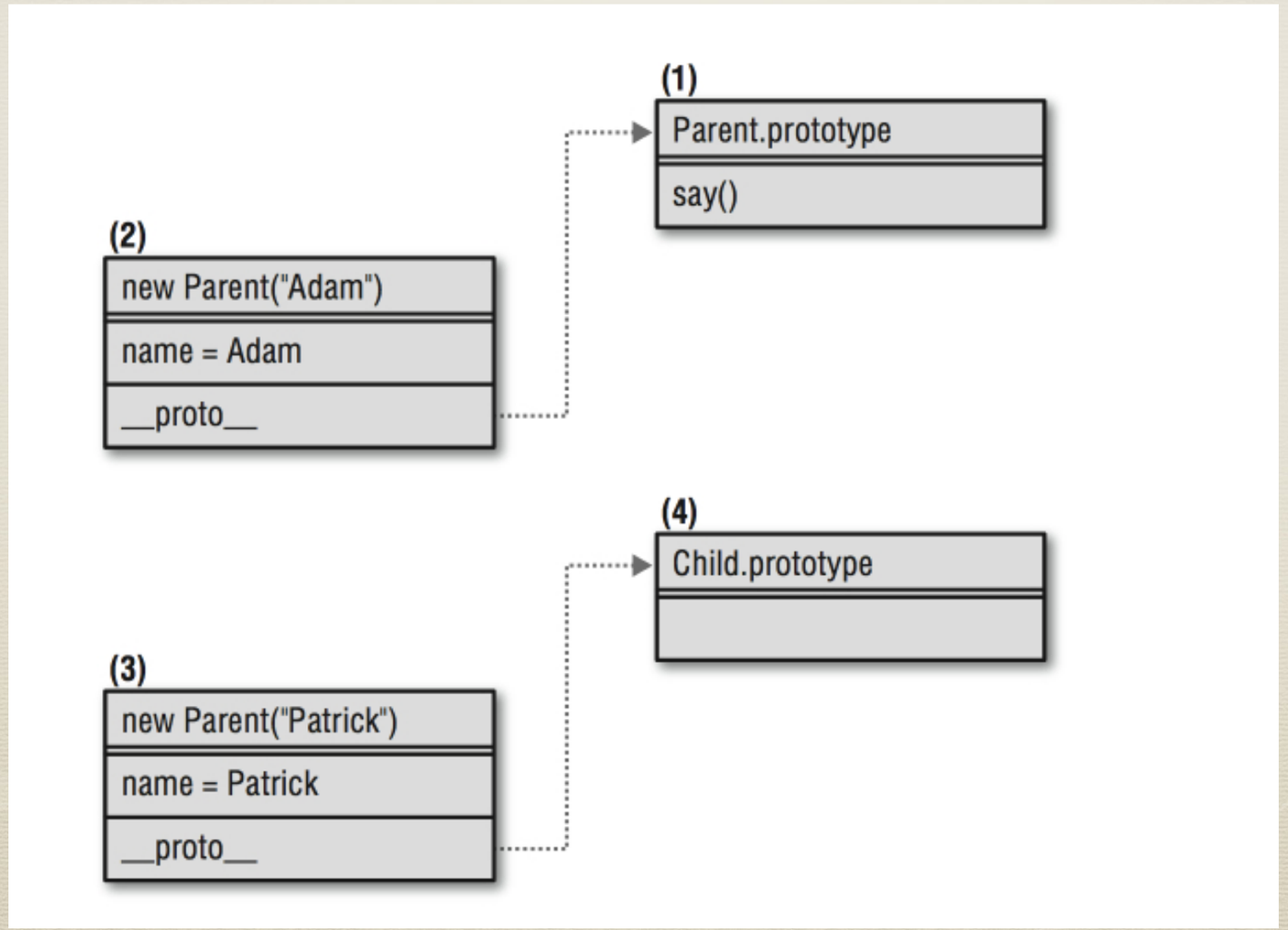
```
function inherit(Parent, Child)
{
    for (var field in Parent) {
        Child[field] = Parent[field];
    }
}
var object = {a : 1};
var object2 = {b: 2};

inherit(object, object2);
console.log(object2.b);
```

Заимствование конструктора

Заимствование конструктора - это шаблон при котором внутри дочернего конструктора вызывается конструктор родительского класса. Таким образом наследуются все свойства объявленые внутри конструктора, но не наследуется prototype.

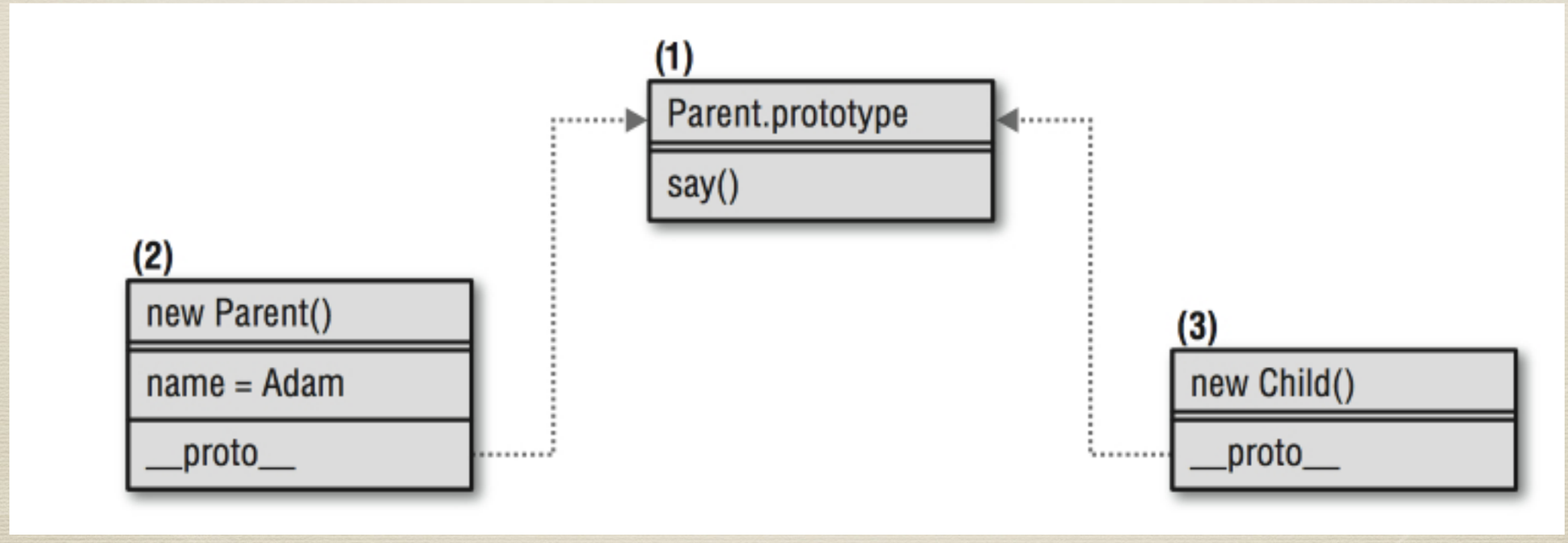
```
function Child(a, c, b, d) {
        Parent.apply(this, arguments);
}
```



Совместное использование прототипов

Шаблон при котором прототипу одного класса присваивается прототип другого класса. Недостаток такого подхода в том что если в дереве наследования будет изменен прототип то он изменится у всего дерева. Также недостатков является то что не наследуются свойства обявленные в конструкторе.

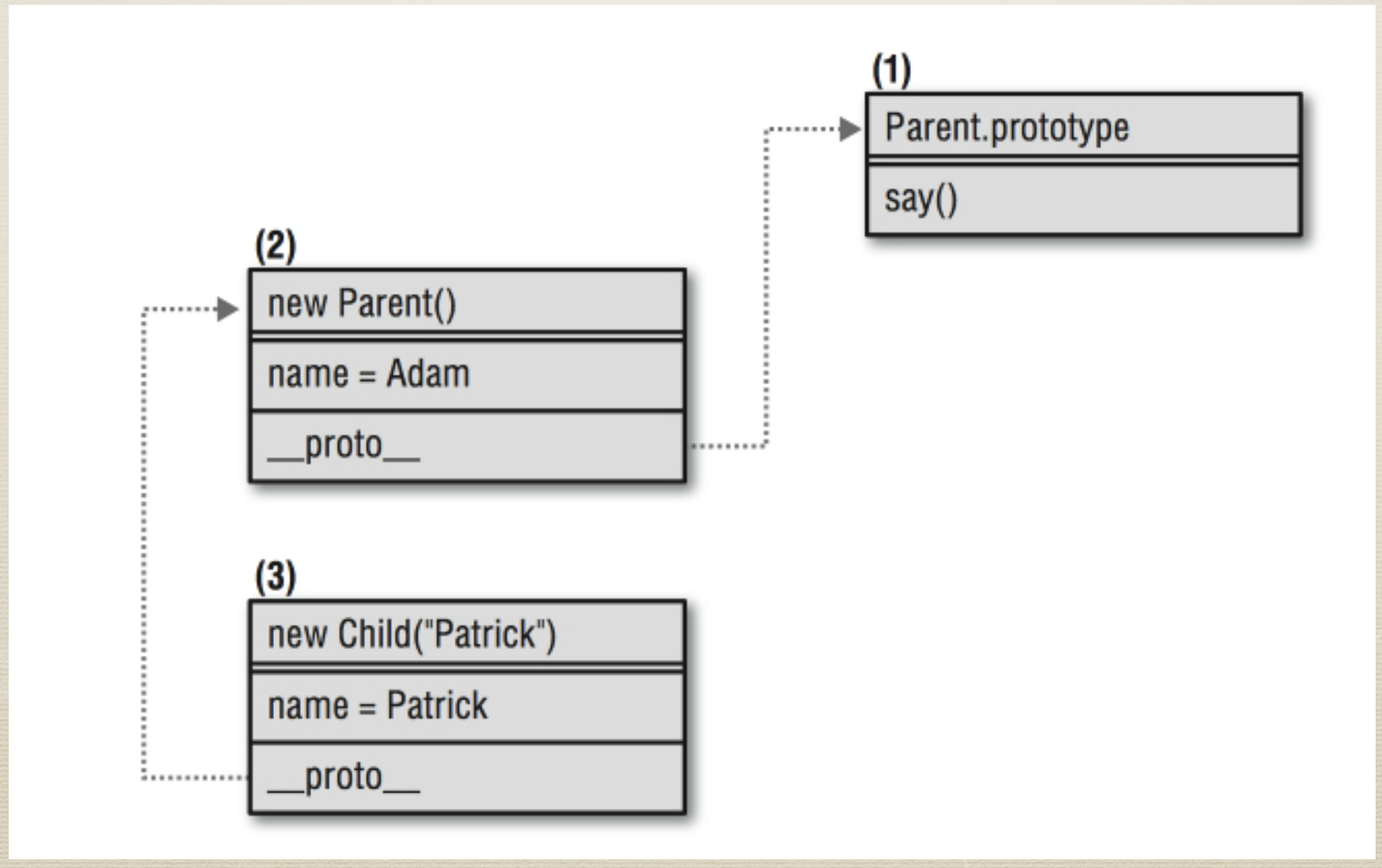
```
function inherit(C, P) {
    C.prototype = P.prototype;
}
```



Смешанное наследование

Такой тип наследования объединяет два варианта наследования - на основе совместного прототипа и заимствования конструктора. Недостаток заключается в необходимости дважды вызывать родительский конструктор, что снижает эффективность

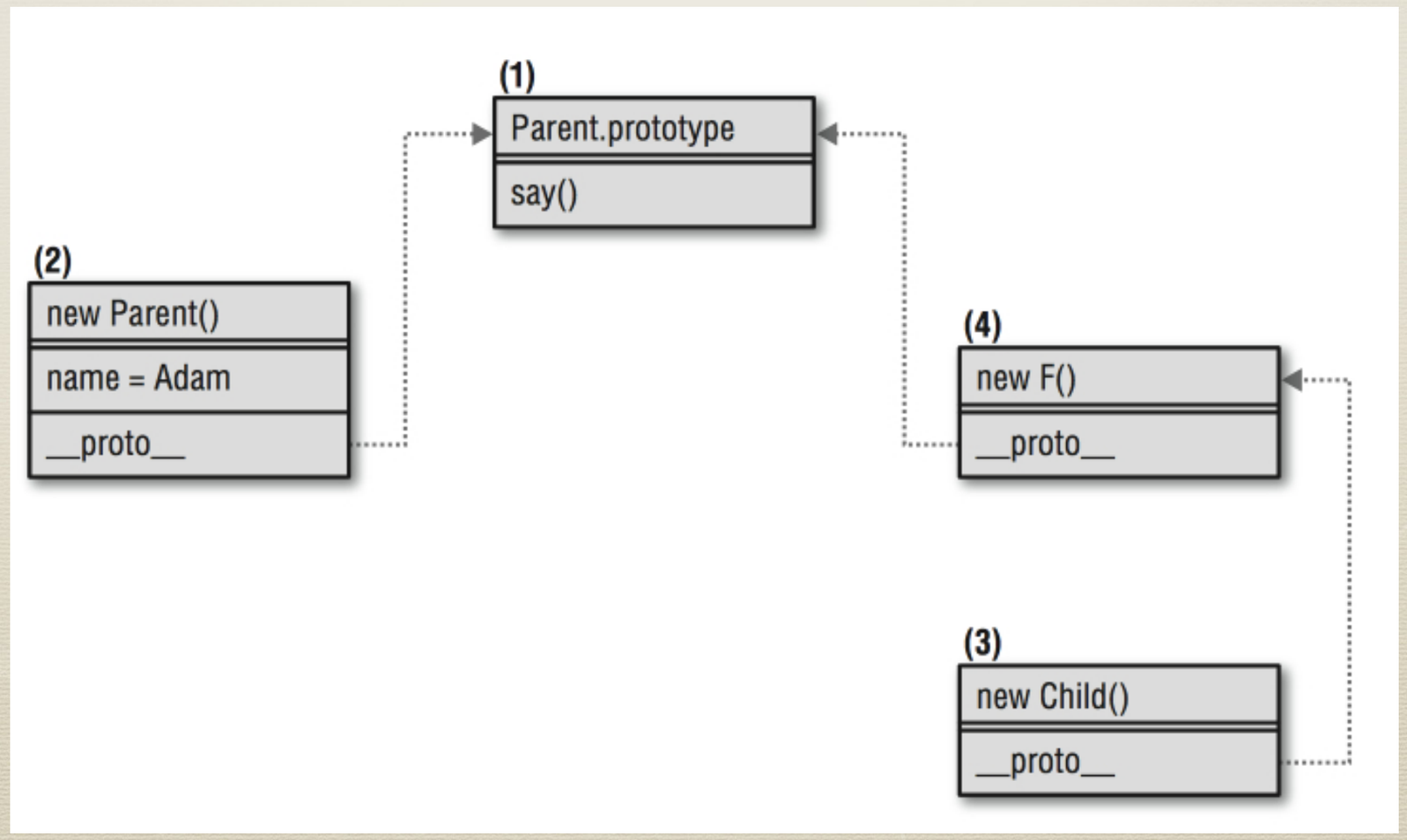
```
function Child(a, c, b, d) {
        Parent.apply(this, arguments);
    }
    Child.prototype = new Parent();
```



Наследование через промежуточный конструктор

Этот шаблон решает проблему, возникающую при совместном использовании прототипа, разрывая прямую связь между родительским и дочерним прототипами, но сохраняя при этом преимущества, которые дает наличие цепочки прототипов

```
function inherit(C, P) {
    var F = function () {};
    F.prototype = P.prototype;
    C.prototype = new F();
}
```



ООП. Статические свойства и методы

Статическими называются свойства и методы, которые не изменяются от экземпляра к экземпляру. JS по умолчанию не поддерживает статические свойства и методы подобно другим языкам программирования. Создание статических свойств и методов осуществляется через добавление свойств и методов к функции конструктору. Статические методы могут быть вызваны только как методы и свойства конструктора и не доступны через this из объектов созданных с помощью этого конструктора.

```
var Class = function () {};
Class.doSomething = function () {};

Class.doSomething();

var instance = new Class();
instance.doSomething();  //error
```

ООП. Приватные методы и свойства классов

Приватными называются методы и классы которые недоступны после создания экземпляра класса, но продолжают существовать и принимают участие в работе объекта. Доступ к таким свойствам осуществляется либо через публичные методы или отсутствует вообще. По умолчанию JS не реализовывает этот механизм. В JS он легко заменяется на механизм замыканий.

```
function Class() {

    // частный член
    var privateProperty = ‘SomeValue’;

    // общедоступная функция
    this.getProp = function () {
        return privateProperty;
    };
}

var instance = new Class();
```