APPMOB - Javascript Prototypes

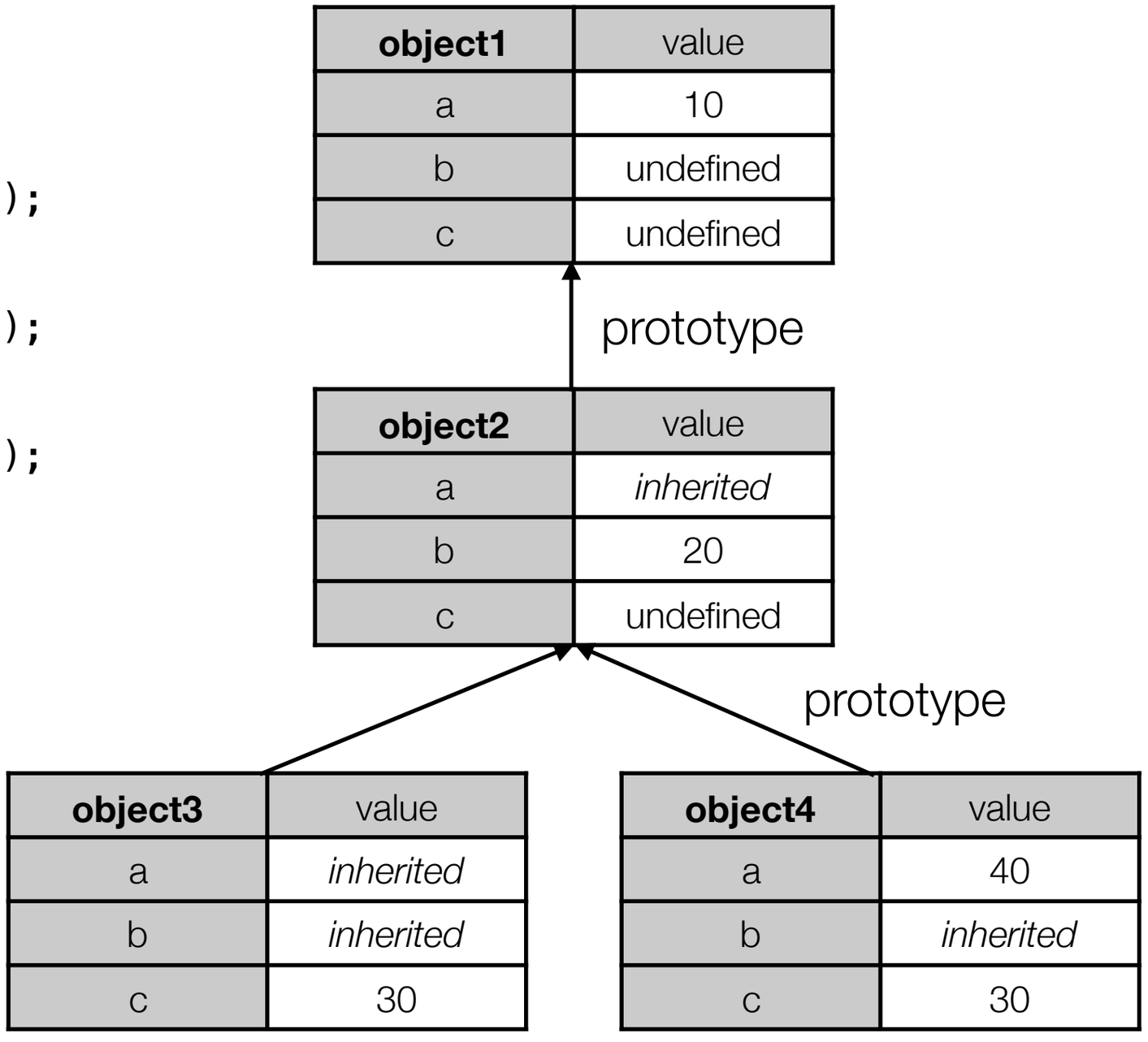
Olivier Liechti & Simon Oulevay
COMEM Applications Mobiles

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Prototypal Inheritance

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

```
var object1 = {};
object1.a = 10;

var object2 = Object.create(object1);
object2.b = 20;

var object3 = Object.create(object2);
object3.c = 30;

var object4 = Object.create(object2);
object4.a = 40;
object4.c = 30;

console.log(object3.a); // 10
console.log(object3.b); // 20
console.log(object3.c); // 30

console.log(object4.a); // 40
console.log(object4.b); // 20
console.log(object4.c); // 30
```



http://codepen.io/AlphaHydrae/pen/bNQKmy/

Class-like Inheritance with Prototypes

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
function Person(name) {
  this.name = name;
}

Person.prototype.isAlive = function() {
  return true;
};

Person.prototype.canEat = function() {
  return true;
};

Person.prototype.hello = function(name) {
  return "Hello " + name + "! I'm " + this.name + ".";
};

var john = new Person("John");

john.isAlive();    // true
john.canEat();     // true
john.hello("Bob"); // "Hello Bob! I'm John."
```

This is a **constructor function**. The **this** variable is the object that will be created. You can attach properties to that object.

The **prototype** of the function will be the prototype of the created object. You can add functions (or methods) to it. All objects created with this constructor function will have these methods (through the prototype).

By using the **new** keyword, you call **Person** as a **constructor function** rather than as a regular function.

Class-like Inheritance with Prototypes

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
function Zombie() {
}

Zombie.prototype = Object.create(Person.prototype);
Zombie.prototype.constructor = Zombie;

Zombie.prototype.isAlive = function() {
  return undefined;
};

Zombie.prototype.hello = function() {
  return "Brainsssssssss!";
};

var graar = new Zombie();

graar.isAlive();    // undefined
graar.canEat();     // true
graar.hello("Bob"); // "Brainsssssssss!"
```

The **prototype** of Zombie a Person. This is how you can simulate class-like behavior.

The Zombie prototype can **override** methods of the Person prototype.

canEat is not defined on the Zombie prototype. It is **inherited** from the Person prototype.

http://codepen.io/AlphaHydrae/pen/RNqBwq/

Class-like Inheritance with Prototypes

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

You usually avoid implementing Javascript "classes" like this, because it's not very readable.

```
function Zombie() {
  Person.call(this, "Zombies have no name");
}

Zombie.prototype = Object.create(Person.prototype);
Zombie.prototype.constructor = Zombie;

Zombie.prototype.hello = function(name) {
  return Person.prototype.call(this, name) + " Let's eat brainsss!";
};
```

This is how you would call the **parent constructor**, the equivalent of **super** in Java.

This is how you would call the method of the **parent prototype**, the equivalent of **super** in Java.

Class Libraries

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Use an existing class library. For example:

http://ejohn.org/blog/simple-javascript-inheritance/

```
var Person = Class.extend({
  init: function(isDancing) {
    this.dancing = isDancing;
  },
  dance: function() {
    return this.dancing;
  }
});

var Ninja = Person.extend({
  init: function() {
    this._super(false);
  },
  dance: function() {
    // Call the inherited version of dance()
    return this._super();
  },
  swingSword: function(){
    return true;
  }
});
```

```
var p = new Person(true);
p.dance(); // true

var n = new Ninja();
n.dance(); // false
n.swingSword(); // true
```