Assignment 1: Multiple Inheritance in Javascript

The Assignment

This assignment is about implementing multiple inheritance in Javascript.

Attention: In this assignment you must not use the built-in **new** operator in Javascript because that would only make things more complicated!

Prototype-Based Multiple Inheritance

Javascript is a prototype based object oriented programming language. You can create new objects by calling 'Object.create(prototype)', where 'prototype' becomes the prototype of the created object.

Your task is to create an object 'myObject' with a 'create' method that supports multiple inheritance of functions: 'myObject.create(prototypeList)', where 'prototypeList' is a list of prototype objects (which may be null or empty). The 'create' method should be inherited by all object created by the 'create' method.

The multiple inheritance should work as follows: when invoking a function of an object that does not exists within the object itself then it is first search for in its first prototype, then in its second prototype etc. When searching for a function in a prototype, if it does not exist within the prototype then it is search for in the prototypes of the prototype etc. Thus the search for a function should be performed in a depth-first manner and from beginning-to-end in each object's list of prototypes.

To be able to control the function lookup ourselves you also need to implement a 'call(funcName, parameters)' method, which should search for a function as described above. The 'call' method takes two parameters: the name of the function 'funcName' and a list of parameters 'parameters' to the function. The 'call' method should be implemented in 'myObject' and should be inherited by all objects created by 'myObject.create(prototypeList)'.

Example:

```
var obj0 = myObject.create(null);
obj0.func = function(arg) { return "func0: " + arg; };
var obj1 = myObject.create([obj0]);
var obj2 = myObject.create([]);
obj2.func = function(arg) { return "func2: " + arg; };
var obj3 = myObject.create([obj1, obj2]);
var result = obj3.call("func", ["hello"]);
```

where 'result' is assigned "func0: hello".

Class-Based Multiple Inheritance

Although Javascript is a prototype based object oriented programming language, you can still create particular objects, which represent classes.

Your task is to create a function 'createClass(className, superClassList)', where 'className' is the name of the created class and 'superClassList' is a list of super-classes the created class should inherit from.

The multiple inheritance should work in a similar way as in the previous task above: when invoking a function of an object that does not exists within the object itself then it is first search for in its class, then in the first superclass of its class, then in the second superclass of its class etc. When searching for a function in a superclass, if it does not exist within the superclass then it is search for in the

superclasses of the supeclass etc. Thus the search for a function should be performed in a depth-first manner and from beginning-to-end in each class's list of superclasses. Note that since Javascript is a dynamic language new functions of an object can be added to the object without involving its class definition.

Every class object should have a function 'new()', which creates an instance object of the class.

Every instance object should have a method 'call(funcName, parameters)', which should search for a function as described above, and invoke the function 'funcName' with the parameters 'parameters'.

Example:

```
var class0 = createClass("Class0", null);
class0.func = function(arg) { return "func0: " + arg; };
var class1 = createClass("Class1", [class0]);
var class2 = createClass("Class2", []);
class2.func = function(arg) { return "func2: " + arg; };
var class3 = createClass("Class3", [class1, class2]);
var obj3 = class3.new();
var result = obj3.call("func", ["hello"]);
```

where 'result' is assigned "func0: hello".

Circular Inheritance Prevention

There is a risk for circular inheritance both with prototype-based and class-based inheritance. To get a grade A or B you must prevent from such circular inheritance in you solutions of both prototype-based multiple inheritance and class-based multiple inheritance.

Grading

Each part, prototype-based multiple inheritance, class-based multiple inheritance and cicular inheritance prevention, will be given a valuation with respect to the quality of the implementation:
a) NotOk means it is missing or not functioning,
b) Ok means it is substantially functioning and the code is ok written and ok structured.
c) Good means it is completely functioning and the code is well written and well structured.

The following table will be used for grading:

Grade	**Prototype-Based**	**Class-Based**	**Circular Prevention**
A	Good	Good	Good
B	Good	Ok	Ok
C	Good	Ok	NotOk
D	Ok	Ok	NotOk
E	Ok	NotOk	NotOk

Good luck!