Packaging and easy OOP concept in JavaScript (Zebra)

What is Zebra ?

Zebra has a small JavaScript library module where simple and easy for understanding packaging and Object Oriented Programming concept is introduced

WEB : http://zebra.gravitysoft.org
e-mail: vish@gravitysoft.org
mobile: +31(0)643400468

Packaging

Add zebra JS library	add to html "header": `<script src='http://repo.gravitysoft.org/zebra/easyoop.min.js' type='text/javascript'></script>`	

Key zebra objects	`zebra`
	`zebra.Class`
	`zebra.Interface`
	`zebra.instanceOf(obj, clazz)`
		- all zebra classes, methods and variables are hosted in "zebra" namespace
		- class is key object to declare any zebra class
		- interface is key object to declare any zebra interface
		- "instanceOf" method has to be used for all declared zebra classes, interfaces and instances of the classes and interfaces as replacement of standard JS "instanceOf" operator

Name spaces	`zebra`
	
	`var ns = zebra.namespace("MY");`
		"zebra" is global variable that represents zebra namespace (to host packages, methods, variables, etc)
		
		define new namespace "MY"

Packaging	`zebra("pkg");`
	`zebra("java.io");`
	
	`zebra.pkg.value = 100;`
	`zebra.java.value = 200;`
	`zebra.java.io.value = 300;`
	`zebra["java.io"].value;`
	`zebra["java"].value;`
	
	`var myNS = zebra.namespace("MY");`
	`myNS("pkg");`
	`MyNS.pkg.value = "test";`
		declare "pkg", "java" and "java.io" packages in zebra name space
		
		any declared package is becoming accessible as a namespace field
		
		define variable "value" in all ("pkg", "java", "java.io") created packages
		
		packages and variables also can be accessible by key
		
		declare new namespace "MY"
		define "pkg" package in new namespace "MY"
		define variable "value" in "pkg" package of "MY" namespace

Import package variables into local name space	`zebra("pkg");`
	`zebra.pkg.value = 10;`
	`eval(zebra.Import("pkg"));`
	`print(value);`
	`// access by full path`
	`zebra.pkg.value++;`
	
	`// store in local variable`
	`var pkg = zebra.pkg;`
	`pkg.value++;`
		all objects defined in the given package can be imported into local name space by applying "eval" method to result of "zebra.Import" method.
		
		it is preferable to use full namespace-package path to an object or store a reference to it in local variable
		
		store an object reference to local variable
		access package variable "value"

Easy Object Oriented Programming concept

Class declaration

```
var A = Class([
     function a() {
       ...
     },
     function b() { ... }
]);
```

declare class *"A"* that defines two public methods: *"a"* and *"b"*

class declaration is an array of methods

Class Instantiation

```
var obj = new A();
obj.a();
obj.b();
```

instantiate declared above class *"A"* and call the class instance methods "a" and "b"

Interface declaration

```
var I = Interface();
```

interface is nothing more than a class marker

Inheritance

```
var B = Class(A, I, [
    ...
]);

var obj = new B();
obj.a();
obj.b();

// all methods executions
// below return true
zebra.instanceOf(obj, A);
zebra.instanceOf(obj, B);
zebra.instanceOf(obj, I);
```

declare class *"B"* that inherits class *"A"* and interface *"I"*

instantiate class *"B"* and call inherited from class *"A"* methods "a" and "b"

use "zebra.instanceOf" method for all zebra declared objects

Constructors

```
var A = Class([
    function(p1) { ... },
    function(p1, p2) { ... }
]);
var obj1 = new A(1);
var obj2 = new A(1,2);
```

constructors are anonymous functions (functions names are not defined)

instantiate two objects and call appropriate constructor

Method overriding

```
var A = Class([
  function a() { ... },
]);

var B = Class(A, [
   function a() { ... }
]);
```

declare class "A" with public method "a"

declare class "B" that inherits class "A" and overrides method "a" of class "A"

Method overloading

```
var A = Class([
function a() {print("A0");},
function a(p1,p2){
   print("A2");
}]);

var obj = new A();
obj.a(1,2);
```

method "a" with empty arguments list is overloaded with method "a" that requires two arguments

method execution prints "A2"

Call parent class method ($super)

```
var A = Class([
   function a(p) { ... }
]);
var B = Class(A, [
  function a(p) {
      this.$super(p);
}]);
```

a parent class method is accessible by calling "this.$super(...)"

method "a" of class "B" calls method "a" implemented by parent class "A"

Call constructor from another constructor

```
var A = Class([
  function() {
      this.$this(p);
  },
  function(p) { ... }
]);
```

constructor with empty parameters list calls more specialized constructor of the same class "A"

this.$this(...)

Inner Classes

```
var A = Class([
   function a() {return 0;}
]);

var obj = new A(10, [
  function a() { return 1;}
]);
obj.a();
```

declare class "A" that defines public method "a"

instantiate inner class basing on class "A" and override method "a" in the new instance

calling method "a" returns 1

Class fields declaration

```
var A = Class([
  function (a) {
     this.a = a;
     this.b = 10;
  }
]);
```

declare class "A" that defines constructor where two fields "a" and "b" are initialized

constructor is the place where all fields have to be declared and initialized

Advanced (alternative) class declaration: static methods and fields, abstract and final methods

```
var A = Class(function($){
 var v = 10;

 this.A = 100;
 this.a = function(p1,p2)
{...}

 $(function a() { ... });

 this.Final(function b(){
    p(); // call private
 });

 this.Abstract(function c(){
    ...
 });

 function p() { ... }
});
```

Declare class A that defines:
- static field "A"
- static method "a"
- public method "a"
- final public method "b"
- abstract method "c"
- private field "v"
- private method "p"

Class "A" cannot be instantiate since it is abstract

Final method "b" of class "A" cannot be overridden

Class "A" has to be inherited by another class that provides an implementation of abstract method "c"