Constructing simply UI classes with Javascript

Constructing simple UI classes with JavaScript

This page is describing how you could make custom UI components with JavaScript and add them into your WRT widget. The example UI is really simple, it is using normal javascript UI components, and includes only static label, button and text box. The original example was written in Dojo and was explained in Dojo: the definitive guide book. The Dojo example shown here is modified version of the original, and all other examples are then based on this modified version.

Standard JavaScript implementation

JavaScript does not really have classes in a normal object oriented way, anyway, you can construct sort-of classes by using the prototype method as shown in following source code:

```
Genie = function(divId){
 this.initialize(divId);
};

Genie.prototype.divId = "genie";
Genie.prototype._predictions = [
                       "As I see it, yet",
                       "Ask again later",
                       "Better not tell you now",
                       "Cannot predict now",
                       "Concentrate and ask again",
                       "Don't count on it",
                       "It is certain",
                       "It is decidedly so",
                       "Most likely",
                       "My reply is no",
                       "My sources say no",
                       "Outlook good",
                       "Outlook not so good",
                       "Reply hazy, try again",
                       "Signs point to yes",
                       "Very doubtful",
                       "Without a doubt",
                       "Yes",
                       "Yes - definitely",
                       "You may rely on it"
                       ];

Genie.prototype._getPrediction = function(){
 var idx = Math.round(Math.random() * 19)
 return Genie.prototype._predictions[idx];
};

Genie.prototype.initialize = function(divId){

 if(divId && divId.length)
  this.divId = divId;

 var label = document.createElement("p");
 label.innerHTML = "Ask a question. The genie knows the answer...";

 var question = document.createElement("input");
```


(C) Copyright Nokia 2014. All rights reserved.

```
   question.size = 50;

  var button = document.createElement("button");
  button.innerHTML = "Ask!";
  button.onclick = function(){
   alert(Genie.prototype._getPrediction());
   question.value = "";
  }

  var container = document.getElementById(this.divId);
  container.appendChild(label);
  container.appendChild(question);
  container.appendChild(button);
 };
```

Then to get the UI component to display in the HTML page, you can use following code:

```
<html>
<head>
 <title>Fun With the Genie!</title>
 <script type="text/javascript" src="js/Genie.js"></script>
 <script type="text/javascript">
  var myGenie = null;

    function myinit(){
   myGenie = new Genie('genie');
  }
 </script>
</head>
<body onload="myinit();">
 <div id="genie"></div>
</body>
</html>
```

If the Genie “class” would be extended with additional methods, you could use the stored myGenie variable for accessing them later on program. The Generic class.zip shows the full WRT example for the implementation.

Dojo implementation

Dojo in sense does not really have classes, but you can use the extend method to construct things that work bit like classes. Dojo has its own way on handling extensions, with the extension implementation, a dojo.provide() line needs to be included to tell the Dojo about the extensions provided by the js file as shown in the following example code:

```
dojo.provide("dtdg.Genie");

 //define the object
 dtdg.Genie = function(divId) {
  this.initialize(divId);
 }
 //and now extend it
 dojo.extend(dtdg.Genie, {

  divId: "genie",
  _predictions : [
   "As I see it, yet",
```


(C) Copyright Nokia 2014. All rights reserved.

```
    "Ask again later",
    "Better not tell you now",
    "Cannot predict now",
    "Concentrate and ask again",
    "Don't count on it",
    "It is certain",
    "It is decidedly so",
    "Most likely",
    "My reply is no",
    "My sources say no",
    "Outlook good",
    "Outlook not so good",
    "Reply hazy, try again",
    "Signs point to yes",
    "Very doubtful",
    "Without a doubt",
    "Yes",
    "Yes - definitely",
    "You may rely on it"
   ],

   _getPrediction : function( ) {
    var idx = Math.round(Math.random( )*19)
    return this._predictions[idx];
   },

   initialize : function(divId) {

    if(divId && divId.length)
     this.divId = divId;

    var label = document.createElement("p");
    label.innerHTML = "Ask a question. The genie knows the answer...";

    var question = document.createElement("input");
    question.size = 50;

    var button = document.createElement("button");
    button.innerHTML = "Ask!";
    button.onclick = function( ) {
     alert(dtdg.Genie.prototype._getPrediction( ));
     question.value = "";
    }

    var container = document.getElementById(this.divId);
    container.appendChild(label);
    container.appendChild(question);
    container.appendChild(button);
   }
  });
```

And then with HTML, Dojo extensions are not included as js files, instead dojo.require() is used to tell the Dojo to look for the extension and to load the required js file(s). Otherwise the HTML is rather much the same as with the generic example:

```
<html>
<head>
```


(C) Copyright Nokia 2014. All rights reserved.

```
 <title>Fun With the Genie!</title>
 <script type="text/javascript" src="dojo/dojo.js"> </script>
 <script type="text/javascript">
  dojo.require("dtdg.Genie");

  var myGenie = null;

    function myinit(){
   myGenie = new dtdg.Genie('genie');
  }

  dojo.addOnLoad(myinit);
 </script>
</head>
<body>
 <div id="genie"></div>
</body>
</html>
```

The Dojo class.zip shows the full WRT example for the implementation.

JQuery implementation

JQuery implementation is nearly identical with the Dojo implementation, with JQuery, there also are no real defined classes, instead the class objects can again be made with the extend method provided by JQuery as shown in the following example:

```
 Genie = function(divId) {
    this.initialize(divId);
 }

 $.extend(Genie.prototype, {
     // object variables
     divId: 'genie',

  _predictions: [
   "As I see it, yet",
   "Ask again later",
   "Better not tell you now",
   "Cannot predict now",
   "Concentrate and ask again",
   "Don't count on it",
   "It is certain",
   "It is decidedly so",
   "Most likely",
   "My reply is no",
   "My sources say no",
   "Outlook good",
   "Outlook not so good",
   "Reply hazy, try again",
   "Signs point to yes",
   "Very doubtful",
   "Without a doubt",
   "Yes",
   "Yes - definitely",
   "You may rely on it"
  ],
```


(C) Copyright Nokia 2014. All rights reserved.

```
  _getPrediction: function( ) {
   var idx = Math.round(Math.random( )*19)
   return this._predictions[idx];
  },
     initialize: function(divId) {

   if(divId && divId.length)
    this.divId = divId;

   var label = document.createElement("p");
   label.innerHTML = "Ask a question. The genie knows the answer...";

   var question = document.createElement("input");
   question.size = 50;

   var button = document.createElement("button");
   button.innerHTML = "Ask!";
   button.onclick = function( ) {
    alert(Genie.prototype._getPrediction( ));
    question.value = "";
   }

   var container = document.getElementById(this.divId);
   container.appendChild(label);
   container.appendChild(question);
   container.appendChild(button);
     }
});
```

As JQuery does not implement any additional extension management, the js file is included in the html as usual, and the HTML otherwise is nearly identical to the Dojo example:

```
<html>
<head>
 <title>Fun With the Genie!</title>
 <script type="text/javascript" src="js/jquery-1.3.2.js"> </script>
 <script type="text/javascript" src="js/jquery.genie.js"> </script>
 <script type="text/javascript">
  var myGenie = null;

    function myinit(){
   myGenie = new Genie('genie');
  }

  $(myinit);
 </script>
</head>
<body>
 <div id="genie"></div>
</body>
</html>
```

The JQuery class.zip shows the full WRT example for the implementation.

(C) Copyright Nokia 2014. All rights reserved.

YUI Version 3 implementation

Similarly as with Dojo & JQuery, YUI also does not really define any base that could be used for forming classes, thus the YUI version 3 implementation is rather similar to the Dojo and JQuery. Note that for some reason Y.extend() appears to work only when combining classes, and as with this example we are adding methods and variable to a “class”, the Y.mix() is used instead of the Y.extend(). Also note that since the interface to YUI APIs is only given as a argument for the callback, and not as a global object, the whole code is wrapped in a one function, and the YUI interface is given to it as a second argument:

```
 Genie = function (divId, Y){

  _Genie = function(){}

  Y.mix(_Genie, {

   divId: "genie",
   _predictions: ["As I see it, yet", "Ask again later", "Better not tell you now",
"Cannot predict now", "Concentrate and ask again", "Don't count on it", "It is certain",
"It is decidedly so", "Most likely", "My reply is no", "My sources say no", "Outlook
good", "Outlook not so good", "Reply hazy, try again", "Signs point to yes", "Very
doubtful", "Without a doubt", "Yes", "Yes - definitely", "You may rely on it"],

   _getPrediction: function(){
    var idx = Math.round(Math.random() * 19)
    return this._predictions[idx];
   },

   initialize: function(divId){

    if (divId && divId.length)
     this.divId = divId;

    var label = document.createElement("p");
    label.innerHTML = "Ask a question. The genie knows the answer...";

    var question = document.createElement("input");
    question.size = 50;

    var button = document.createElement("button");
    button.innerHTML = "Ask!";
    button.onclick = function(){
     alert(_Genie._getPrediction());
     question.value = "";
    }

    var container = document.getElementById(this.divId);
    container.appendChild(label);
    container.appendChild(question);
    container.appendChild(button);
   }
  });

  _Genie.initialize(divId);
  return _Genie;
 }
```

The HTML then again looks nearly identical to the previous examples, only real difference is that the init function now also gets

(C) Copyright Nokia 2014. All rights reserved.

the YUI API interface as an argument:

```
<html>
<head>
 <title>Fun With the Genie!</title>
 <script type="text/javascript" src="js/yui-min.js"></script>
 <script type="text/javascript" src="js/Genie.js"></script>
 <script type="text/javascript">
  var myGenie = null;

  function myinit(Y){
   myGenie = new Genie('genie',Y);
  }

   YUI().use('node', myinit);
 </script>
</head>
<body>
 <div id="genie"></div>
</body>
</html>
```

The YUI3 class.zip shows the full WRT example for the implementation.

Mootools implementation

Mootools is the only one of these five different methods that actually defines a Class with inheritance. Anyway, the implementation is basically nearly identical to other ones, just the wrapping is different. With Motools I also decided to inherit the Options with Implements, to get the div id handling done in native way.

```
var Genie = new Class({
 Implements: Options,
 options: {
  divId: 'genie'
 },

 _predictions: [
   "As I see it, yet",
   "Ask again later",
   "Better not tell you now",
   "Cannot predict now",
   "Concentrate and ask again",
   "Don't count on it",
   "It is certain",
   "It is decidedly so",
   "Most likely",
   "My reply is no",
   "My sources say no",
   "Outlook good",
   "Outlook not so good",
   "Reply hazy, try again",
   "Signs point to yes",
   "Very doubtful",
   "Without a doubt",
   "Yes",
   "Yes - definitely",
```


(C) Copyright Nokia 2014. All rights reserved.

```
    "You may rely on it"
  ],

  _getPrediction: function () {
   var idx = Math.round(Math.random( )*19)
   return this._predictions[idx];
  },

  initialize: function(options){

   this.setOptions(options);

   var label = document.createElement("p");
   label.innerHTML = "Ask a question. The genie knows the answer...";

   var question = document.createElement("input");
   question.size = 50;

   var button = document.createElement("button");
   button.innerHTML = "Ask!";
   button.onclick = function( ) {
    alert(Genie.prototype._getPrediction( ));
    question.value = "";
   }

   var container = document.getElementById(this.options.divId);
   container.appendChild(label);
   container.appendChild(question);
   container.appendChild(button);
   }
});
```

The HTML is again nearly identical, the only real difference to previous examples is the argument format passed for the Genie:

```
<html>
<head>
 <title>Fun With the Genie!</title>
 <script type="text/javascript" src="js/mootools-1.2.4-core.js"></script>
 <script type="text/javascript" src="js/Genie.js"></script>
 <script type="text/javascript">
  var myGenie = null;

    function myinit(){
   myGenie = new Genie({div: 'genie'});
  }

   window.addEvent('domready',myinit);
 </script>
</head>
<body>
 <div id="genie"></div>
</body>
</html>
```

The Mootools class.zip shows the full WRT example for the implementation.

(C) Copyright Nokia 2014. All rights reserved.

(C) Copyright Nokia 2014. All rights reserved.