

## COMP 322 Internet Systems Fall 2007 Assignment 3

Due in the digital drop box on Monday, Oct. 1 by 11:00 PM

1. This problem involves the JavaScript object model. You will define constructor functions `Faculty` and `Course`. An instance of `Course` will have an instance of `Faculty` as the value of its `instructor` instance variable. You can download (from where you got this assignment) a file `testProbl.js` with JavaScript test code and an `.html` file, `testProbl.html`, that accesses the two files you write (`faculty.js` and `course.js`) as well as `testProbl.js`. Each of the files you write contains only a constructor function and should have no more than about 15 lines and possibly as few as ten lines. Immediately below, we specify the two constructor functions in detail.

Constructor function `Faculty` has arguments `name` and `dept`. Likewise, it has instance variables `name` (a string, with default “name unknown”) and `dept` (a string, with default “department unknown”). Write `get_` and `set_` methods for both instance variables. The `toString()` method should return strings of the form “<name> of <dept>”, where <name> is the value of instance variable `name` and <dept> is the value of instance variable `dept`.

Constructor function `Course` has arguments `name`, `times`, `instructorName`, and `instructorDept`, all expected to have string values. The instance variables are `name` (a string, with default “unknown course name”), `times` (a string, with default “unknown times”), and `instructor` (instance of `Faculty`, with no default). The instance of `Faculty` must be constructed using `new` and the `Faculty` constructor, passing it the values of the last two arguments to this constructor. Define `get_` and `set_` methods for all instance variables. Note that `setinstructor()` takes an instance of `Faculty` as its parameter, and `getinstructor()` returns an instance of `Faculty`. The `toString()` method should just return the string that is the value of the `name` instance variable.

The following is the listing of the test file.

```
var fac1 = new Faculty( "Dr. Smith", "Math" );
document.writeln( "New faculty: " + fac1.getname() + ", " +
fac1.getdept() + "<br>" );
document.writeln( "Again: " + fac1 + "<br><br>" );

var fac2 = new Faculty();
document.writeln( "New faculty: " + fac2.getname() +
                    ", " + fac2.getdept() + "<br>" );
fac2.setname( "Dr. Jones" );
fac2.setdept( "CS" );
document.writeln( "Updated: " + fac2.getname() +
                    " " + fac2.getdept() + "<br><br>" );

var crs1 = new Course( "COMP322", "MWF 1:00-1:50",
                    "Dr. Green", "CS" );
document.writeln( "New course: " + crs1.getname() +
                    ", " + crs1.gettimes() +
                    ", " + crs1.getinstructor() + "<br>" );
document.writeln( "Again: " + crs1 + "<br><br>" );
```

Continued

Continued from previous page

```
var crs2 = new Course();
document.writeln( "New course: " + crs2.getname() +
                  ", " + crs2.gettimes() +
                  ", " + crs2.getinstructor() + "<br><br>" );

crs2.setname( "COMP590" );
crs2.settimes( "TR 1:00-2:15" );

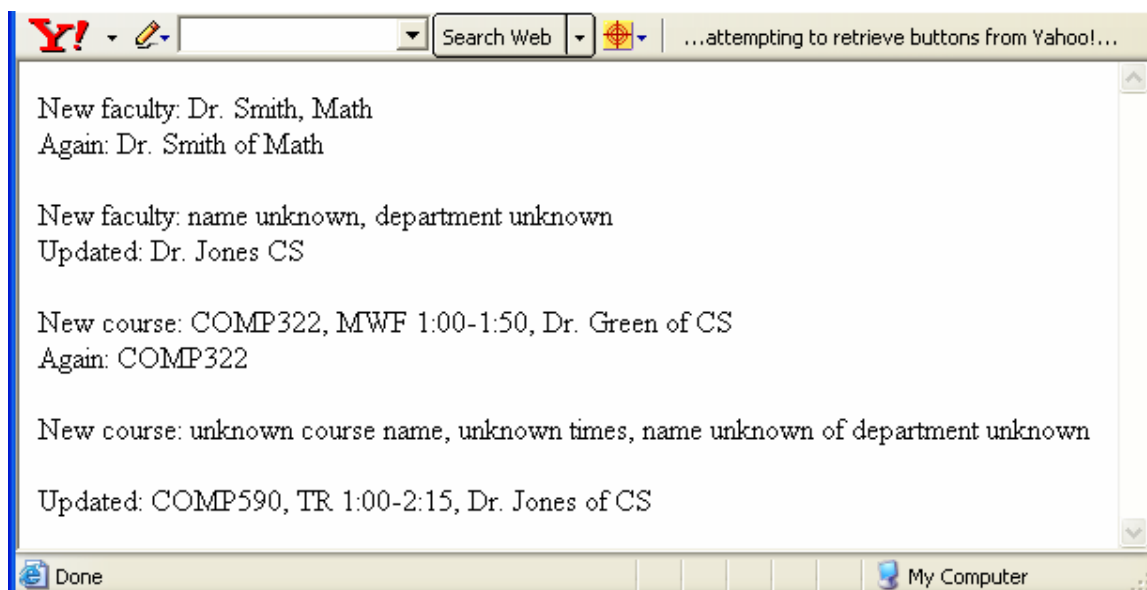
crs2.setinstructor( fac2 );

document.writeln( "Updated: " + crs2.getname() +
                  ", " + crs2.gettimes() +
                  ", " + crs2.getinstructor() );
```

The following is the HTML driver.

```
<html>
<head>
  <title>Assignment 3, Problem 3</title>
  <script type="text/javascript" src="faculty.js">
  </script>
  <script type="text/javascript" src="course.js">
  </script>
  <script type="text/javascript" src="testProbl.js">
  </script>
</head>
</html>
```

The following is the output produced.



2. For this problem, you add an instance method `length` to the `List` prototype that is defined in the class notes. If `lst` is an instance of `List`, then `lst.length()` should return the number of nodes (instances of `Node`) in list `lst`. The file `node.js`, with the definition of the constructor function `ListNode`, does not change—it is just as in the class notes and is available as an example from the class Web site (<http://www.ncat.edu/~esterlin/c322f04/Examples/Part10/node.js>).

The file `list.js`, with the definition of the prototype `List`, is as in the class notes except that it includes the definition of the instance method `length`. You can download this file from the site where you accessed this assignment. There are gaps where you must supply missing code in the definition of `length`. You can also download the file `testProb2.js`, which contains test code, and `testProb2.html`, which is the HTML document that accesses the three `.js` files. The following is the listing of `testProb2.html`:

```
<html>
<head>
<title>List Tests</title>
<script type = "text/javascript" src = "node.js">
</script>
<script type = "text/javascript" src = "list.js">
</script>
<script type = "text/javascript" src = "testProb2.js">
</script>
</head>
</html>
```

The following is the listing of `testProb2.js`:

```
var lst = new List( "test" );
document.writeln( "<p>Initially: " +
                  lst.length() + "</p>" );
lst.insertAtFront( 2 );
lst.insertAtBack( 3 );
lst.insertAtFront( 1 );
document.writeln( "<p>After 3 insertions: " +
                  lst.length() + "</p>" );
```

The rendering that results when `testProb2.html` is executed is shown at right.

Initially: 0

After 3 insertions: 3

The listing of the file `list.js` is shown on the next page. We have left out the code for the functions `List_insertAtFront`, `List_insertAtBack`, `List_removeFromFront`, `List_removeFromBack`, and `List_toString`, which is as in the class notes. There is missing code in the definition of the function `List_length` and for the assignment of this function to the `length` property of `List`'s prototype. Here these gaps are labeled with Greek letters, and descriptions of the missing fragments (identified by these labels) are given after the listing.

```

function List( name )
{
  this.name = name || "a list";
  this.firstNode = this.lastNode = null;
  this.getname =
    function() { return this.name; };
  this.setname =
    function( name ) { this.name = name; };
  this.isEmpty =
    function() { return this.firstNode == null; };
}

function List_length()
{
  var count = 0,
      p = this.firstNode;

  α
  _____
  _____
  _____
  -
  return count;
}

/*
  The definitions of List_insertAtFront, List_insertAtBack,
  List_removeFromFront, List_toString are just as in the
  class notes.
*/

β
_____

List.prototype.insertAtFront
  = List_insertAtFront;
List.prototype.insertAtBack
  = List_insertAtBack;
List.prototype.removeFromFront
  = List_removeFromFront;
List.prototype.removeFromBack
  = List_removeFromBack;
List.prototype.toString
  = List_toString;

```

$\alpha$ : Write a loop that, on each iteration, advances the reference `p` to point to the next node in the list and increments `count`. The loop is exited when `p` runs off the end of the list.

$\beta$ : Assign the function `List_length` to the `length` property of `List`'s prototype.

Turn in all four files (three `.js` files and one `.html` file), even though you write code only for `list.js` (downloaded as `listGaps.js`).

3. The following code (with fragments missing) implements a prototype object `doubleQ`. A `doubleQ` object has two instance variables: `firstQ` and `secondQ`. These are arrays that together implement a queue. The `enqueue()` instance method of `doubleQ` adds its argument to the end of `this.secondQ`. The `dequeue()` instance method removes and returns the element at the front of `this.firstQ`. The final instance method is `transfer()`, which removes the element from the front of `this.secondQ` and adds it to the end of `this.firstQ`. If, however, `this.secondQ` is empty, it does nothing. Test code is included in the file. The output produced by the completed code is shown after the listing. You can download this file (with fragments missing) from the location where you got this assignment.

**Hint:** Use the `push()` and `shift()` Array instance methods.

```
<html>
<head>
  <title>Assignment 3, Problem 3</title>
<script type="text/Javascript">
  function doubleQ( q1, q2 )
  {
    this.firstQ = q1 || [];
    this.secondQ = q2 || [];
    this.enqueue = function( item )
    {
      }; // Code missing
    this.dequeue = function()
    {
      }; // Code missing
    this.toString = function()
    { return "[" + this.firstQ + "] [" +
      + this.secondQ + "]; };
  }

  function doubleQ_transfer()
  {
    // Code missing
  }
  doubleQ.prototype.transfer = doubleQ_transfer;

  var dQ = new doubleQ( [1, 2], [3, 4] );

  document.write( "<p>dQ: " + dQ + "</p>" );
  dQ.transfer()
  document.write( "<p>After transfer: " + dQ + "</p>" );
  document.write( "<p>Value dequeued: " + dQ.dequeue() + "</p>" );
  document.write( "<p>After dequeue: " + dQ + "</p>" );
  dQ.enqueue(5);
  document.write( "<p>After enqueueing 5: " + dQ + "</p>" );
</script>
</head>
<body>
</body>
</html>
```

dQ: [1,2] [3,4]

After transfer: [1,2,3] [4]

Value dequeued: 1

After dequeue: [2,3] [4]

After enqueueing 5: [2,3] [4,5]