JavaScript Advanced Homework – Slide Show

Overview / Requirements

In a previous homework, you wrote a reusable JavaScript “Make” function that returned a simple visual JS object with (public and private) properties and methods and some built-in event handling. You also wrote a (non-reusable) component perhaps called “twoObjects” that invoked the “Make” function twice and returned a DOM element holding two of the visual objects. To your nav bar, you added a link (associated in the routing table with “twoObjects”). When the user clicked on that link, the user was able to see and interact with the two visual objects that had been injected into the content area. (I am describing what you already did so that you can see the similarities to what you are being asked to do in this homework.)

In this homework, you shall write a “MakeSlideShow” function that returns a more sophisticated visual JS object, a slide show, controlled by left/right clickable buttons, that displays images from an array of objects. As before, you shall also write a (non-reusable) component function (perhaps called “SlideShows”) that returns a DOM element holding 2 or 3 of the Slide Show objects. You shall add a “SlideShow” link under the Search Drop Down Menu header. Again, you’ll add a link (associated in the routing table with “SlideShows”). When the user clicks on that link, routing will invoke SlideShows to inject the slide show visual objects into the content area of the page.

Your “SlideShows” component shall read the data for your images and captions from JSON files, using a technique called AJAX. Here is an example of how your page might look after the user clicks on “Search – Slide Shows” (although your version shall also show captions for each image).

Note:
• This homework is about learning JavaScript, so you MAY NOT use jQuery.
• For this particular homework, a lot of things are specified so similar code is a bit more likely than in previous homeworks, but try to change things up whatever you can and add your own twist to it.
Before You Start

In the Tutorial’s JavaScript page (on my website), there is a section about Slideshow. Read the code explanations and locate the slide show code the zip file. In the slide show folder there are several numbered examples:

- The first three examples (01, 02, 03) show how to make a single slideshow with some “hard coding” of HTML code. This code is not reusable – you could not easily make more than one slide show like this but it does show you the elements that go into making a simple slide show.
- Then there is an example (04) of a reusable “MakeSlideShow” (returns a DOM element slide show that can be injected onto a page). This can be used to place any number of slide shows on a page. You’ll definitely want to model your code after this example.
- Example 05 gives an example of how you can layout multiple slide shows on a page.
- You can skip 08 Make Slide Show Auto – that was just for fun. You would not be able to count that as your individual modification (since it was provided in sample code).
- Example 10 shows how to use an AJAX call to read JSON data, convert the JSON to a Java Script object and pass that data to the Make Slide Show function.

Design Specifications

Function MakeSlideShow shall:

- Accept an input parameter object with:
  - (required property) array of objects (each object having an image property and a caption property). If this property is not supplied, MakeSlideShow shall return a DOM element with a nice error message in it (like cannot create slide show – did not receive expected property 'objArray'). You can assume each element in this object array has two properties: image and caption.
  - (optional property): the name of the CSS class that should be applied (by MakeSlideShow) to the slide show DOM element. If this property is not supplied, assume the class name is “slideShow”.
  - At least one optional property that affects how the returned slide show object looks or works. If the property is not provided, the code shall set a default value. This property shall be named according to its purpose and the default value selected accordingly.

Note: we are passing all of this in so that MakeSlideShow makes NO ASSUMPTIONS about anything regarding the HTML page or CSS code. On the next page, there is an example showing how MakeSlideShow might be called, as you are working on MakeSlideShow, before you add in the AJAX etc.

- Return a DOM element (e.g., a “div”) that contains a slide show component with an image and a caption and left and right buttons. When the left button is clicked, the previous image (and caption) shall be shown (unless the image being shown is the first image in the list). When the right button is clicked, the next image (and caption) shall be shown (unless the image being shown is the last image in the list).
  - The DOM element returned by MakeSlideShow shall have at least one public method which affects how the slideshow looks or works. In the sample code, the DOM element returned by my MakeSlideShow function had a public method that allowed the setting of the initial picture number. There was another example that provided a public method that made the slide show automatic, with specified time interval between images (I think this version might “roll around” to the first image after advancing to the last). You have to come up with a different public method, hopefully something unique.
- Not make any assumptions about any code outside itself (other than what’s passed as input parameter), but it is OK for MakeSlideShow to assign “caption” as a style
Here is an example of how you might invoke MakeSlideShow as you are working on it, before you get to adding in AJAX etc. I named the properties “not caption” and “not image” to show you how you can deal with that (in case the JSON file did not happen to have those properties).

```javascript
var myObjArray = [ // hard coding sample data into a JS array of objects, before adding AJAX...
    {
        "item": "Jet Ski",
        "pic": "http://cis-linux2.temple.edu/~sallyk/pics_waterFun_jpg/jetski.jpg",
        "price": "5000"
    },
    {
        "item": "Ski Boat",
        "pic": "http://cis-linux2.temple.edu/~sallyk/pics_waterFun_jpg/skiBoat.jpg",
        "price": "50000"
    }
];

// since the objects in your array do not have properties image and caption, create these properties:
for (var i = 1; i < myObjArray.length; i++) {
    myObjArray[i].image = myObjArray[i].pic;
    myObjArray[i].caption = myObjArray[i].item;
}

var ss1 = MakeSlideShow( {
    objArray: myObjArray, // required, contains array of objects each having an "image" and a "caption" property.
    captionStyleName: "captionClass", // CSS class name to assign to the caption div within the slide show. // Look for grey highlight in CSS code on next page.
    optionalPreference: "blue" // example of optional property - but pick a more representative // name than "optionalPreference"
} );

Here's an example of how (in MakeSlideShow) you could check for optional property of input parameter object:

```javascript
function MakeSlideShow(params) {
    var optionalPreference = params.optionalPreference || "green";
}
```

Do not use “optionalPreference” as your property name. Give it a name that represents whatever you have chosen for homework.

**Function ajax** shall:

- Take the following input parameters (or you can modify function ajax to take a parameter object with the following properties):
  - URL to invoke (for the AJAX call),
  - a success function (a function to be called back in case of AJAX success) and
  - a DOM element (where an error message will be written in case of AJAX failure).
- not make any assumptions about any code outside itself (other than what’s passed as input parameter).
Function SlideShows shall:

- Take no input parameters (so that it can be invoked by our routing Provider code).
- Read at least two JSON files that each get converted to a JS object array.
- To be realistic, at least one of your object arrays shall not have “image” and “caption” properties. In other words, don’t just edit the JSON files to be as you wish them to be. Pretend you are getting this data from a 3rd party and you don’t have control over what it is. Use a loop (as shown above) to create the necessary properties (“image” and “caption”) that MakeSlideShow expects.
- Invoke MakeSlideShow once for each JSON file, passing to it the corresponding JS object array.
- In one invocation supply a value for your optionalPreference (whatever you may have named it). In the another invocation, make sure to NOT supply that property (MakeSlideShows should supply a default value).
- For each invocation of MakeSlideShow, an object is returned. That object has at least one public method (of your choosing). SlideShows shall invoke that public method for at least two of the objects returned.
- Return a DOM element that contains the slide show visual objects created by invoking MakeSlideShow.
- Function SlideShows CAN make assumptions about the HTML page: JSON file names, CSS Style name, etc. This is NOT a reusable function. The purpose of function SlideShows is to layout the content as the HTML coder wishes it to look (whenever the user clicks on a certain link from the nav bar).
- NOTE: as SOON as you start using AJAX, you must RUN, not VIEW your page. AJAX needs a web server to respond to the HTTP request.

Each of the above three functions (MakeSlideShow, ajax, and SlideShows) shall:

- Be in an external JS file named the same as the function.
- Be the only code in the JS file.
- Remember that for every external JS file you want to use, the HTML page needs a <script> tag referencing it.

Your HTML page shall:

- have (as its first line of JavaScript code) the "use strict" directive as its first line of code (see below). By having this directive, if you inadvertently declare a new variable by misspelling a variable name, you’ll throw a JavaScript exception (easier to debug).

"use strict";
Styling

Obviously, a good slide show component is going to need some styling. To handle that, the SlideShow Provider code should provide default styling that works well, in an external style sheet. This enables the HTML coder to override whatever styling choices were made or use what was provided. Clearly, this gives the HTML coder the opportunity to visually “ruin” the beautiful slideshow you have created, but you did provide them with a perfectly good style sheet that they could either use or improve upon (to their taste).

Your Slide Show external style sheet shall contain styles for a single (global) classname (e.g., “slideShow” in the example below). All other selectors shall be compound selectors that style subcomponents of that classname (see example):

```css
.slideShow { /* styling the slide show probably div */
    text-align:center;
    font-weight:bold;
}
.slideShow img { /* style images that are inside of class slideShow */
    width: 25%;
    border: 3px inset graytext;
    border-radius:8px;
}
.slideShow .captionClass { /* style for elements classed “captionClass” but only if they are inside of class slideShow */
    padding-bottom: 1ex; /* captionClass is grey because it was referenced earlier – look for it above */
}
.slideShow button { /* style for any button that is inside of class slideShow */
    width: 50px;
    margin-right: 10px;
    margin-left: 10px;
}
```
JSON Files and Images

To save you some time, I have prepared several JSON files that you may use. Click on any of these links to download a local copy to save to your web site. These JSON files reference images (all the same dimensions and) already published on my web site. Because these images are all the same size, the slide show will look good, not grow/shrink each time the user advances to the next image.

- http://cis-linux2.temple.edu/~sallyk/pics_car/cars.json
- http://cis-linux2.temple.edu/~sallyk/pics_cat/cats.json
- http://cis-linux2.temple.edu/~sallyk/pics_users/users.json
- http://cis-linux2.temple.edu/~sallyk/pics_waterFun_jpg/waterFun.json

Here is an example of what you’ll find in those JSON files:

```json
[
    {
        "item": "Jet Ski",
        "image": "http://cis-linux2.temple.edu/~sallyk/pics_waterFun_jpg/jetski.jpg",
        "price": "5000"
    },
    ...
    {
        "item": "Ski Boat",
        "image": "http://cis-linux2.temple.edu/~sallyk/pics_waterFun_jpg/skiBoat.jpg",
        "price": "50000"
    }
]
```

If you want to find and use your own images and your create your own JSON file, that is fine (click on Tutorials – Web Images for help). However, I recommend that you complete the homework first using the provided images, then come back and add your own images later, time permitting.

- Points WILL NOT be deducted if your image size changes as you advance thru the images.
- Points WILL BE deducted if your image file sizes are so large that they render slowly upon first visit.

Originality and Other Requirements:

1. **Originality.** A lot of this homework is prescribed but there are a few areas where you are asked to come up with your own ideas (optional properties of input parameter to MakeSlideShow and public method of the slideshow object). **Points will be deducted if you submit work that is overly similar to sample code provided from my web site – or to any classmate’s.**

2. **Look and Feel.** Style the slide show (in your role as HTML coder) so that it blends nicely with your page layout.

3. **Blog.** Add a blog entry (header and paragraph) where you describe what you did in this homework, what you found easy, what was hard/confusing, what was valuable.

4. **Good Programming Style.** At the top of the 3344 labs page, there is an entry entitled "Requirements For All Labs". Make sure you adhere to the "good programming and design practices" listed there.
Project Organization

Your project shall be organized as shown, with folders for icons, json, pictures (not shown), style, and, js (reusable/provider style versus components written by HTML coder).

Tips

1. Carefully study the sample code and related explanations if you are “lost” getting started.
2. Always use the NetBeans menu option: Source – Format to keep your code properly indented so that you can understand the nesting of your own code. When you click on a starting brace (or HTML tag), NetBeans should highlight (yellow) that brace and its matching ending brace (or HTML tag).
3. Pay attention to (and get rid of) the red bubbles that NetBeans places to the left of any line of code that has a syntax error. Hover over the red bubble to get an explanation of the error.
4. As you work on this lab (or any lab that involves JavaScript), RUN (not VIEW) your pages in Chrome – this is because AJAX relies on a web server (like Apache or Glassfish) to respond to the HTTP request.
5. In Chrome, press F12, and watch the console tab - this is the only place that you'll see JavaScript runtime error messages. Add console.log() statements to your code so that you can see them printed out in the console. To see what’s in an object console.log(someObject), but do not console log a string concatenated with the object or you will not be able to open up the object properties in Chrome’s console.
6. As with ANY programming (and especially if you are new to JavaScript), code just a few lines of code between testing. That way, if something goes wrong, you’ll know which lines of code to fix.
7. Backup your code (copy the project folder outside of NetBeans) when you get something working (give the copied folder a good representative name).

Submission:

1. After completing all requirements, test locally (and syntax check). Don’t forget to add the blog entry to your blog content area. Then publish, then test what you published (to your “regular” folder, not the LA folder).
2. Perform a self assessment by looking at the deductions (see next page).
3. Then submit your self assessment and a zip file of your whole project into Canvas. The project should include all the Homeworks you have done (all working and in the same project).
Example Deductions:

- -9 if no zip file submitted into Canvas.
- -9 if page not published.
- -1: Missing SlideShow link under the Search Drop Down Menu of the nav bar.
- -3: routing not used to invoke your slide show.
- -1: layout of slide show does not blend in nicely with overall alayout of your page and/or you have not modified any of the slide show styling that was provided as sample.
- Up to -2: you do not have a separate external style sheet devoted to slide show, with nothing but “localized selectors”, meaning all selectors start with the same CSS classname (e.g., “.slideshow”). By placing all of your CSS rules under a single global CSS classname, you avoid polluting the global CSS namespace.
- -7 if there’s only one slide show on your page, instead of the required 2 or more. With two or more slide shows functioning properly, we are more sure that your code was written properly -- object oriented using dependency injection.
- -4 if there is no image as part of the slide show object (e.g., only the caption changes)
- -3 if there is no caption as part of the slide show (e.g., only the image changes)
- -3: Function SlideShows does not use AJAX to read JSON files (perhaps you just copy/pasted the array of objects into your code as was shown in the HW writeup).
- -1: Each of the JSON files included “image” and “caption” as properties. You were asked to pretend the JSON was from a 3rd party meaning you would not control its format. To fix this you should have used a loop to create the necessary properties (as was shown in the HW writeup).
- Up to -2: did not demonstrate optional property of input parameter object (when invoking MakeSlideShow from function SlideShows) and/or how your optional property (and related functionality) does not make sense within the slide show context. From function SlideShows, you were supposed to call MakeSlideShow once with the optionalPreference property supplied, and once not (so that allow MakeSlideShow would provide a default value).
  - -0.5: used the same property name as mentioned in this writeup (“optionalPreference”) for your optional property of the parameter object in MakeSlideShows. You should have given this property a name that’s representative of what you chose for your functionality.
- Up to -3: in SlideShows, you did not invoke the public method of the objects returned by MakeSlideShow and/or MakeSlideShow did not add a public method to the object that it returns.
- Up to -2: Violation of dependency injection meaning that your Provider style code (functions ajax or MakeSlideShow) referenced something that was not passed to it as input parameter. Note: Function SlideShows is not provider style code. It is code written by the HTML coder showing what they want to show on the page. Therefore, function SlideShows IS ALLOWED to reference anything from the web root (like JSON file names, CSS code, etc).
- Up to -2 if your page loads too slowly – try to keep each image around 500K or less.
- Up to -3 for lack of originality, if your code is too similar to sample code. If it is too similar to a classmate, consequences can be more.
- Up to -2 for HTML/CSS syntax errors which show in red font from Firefox View Source.
- Up to -2 for poor coding style. Your code should be neat, with some comments, properly indented, well named, and with no unused code.
- Up to -2 for missing or incomplete self assessment.