First JavaScript Homework:
Drop Down Menus and (Do-It-Yourself) Routing

CONTENTS

1. OVERVIEW.......................................................................................................................... 2
2. WHAT IS A DROP DOWN MENU?.......................................................................................... 2
3. BEFORE YOU START WORKING ON THE DROP DOWN MENU.............................................. 2
4. DROP DOWN REQUIREMENTS AND DESIGN SPECS ............................................................... 3
5. HINTS FOR DROP DOWN SOLUTION ................................................................................... 4
6. BEFORE YOU START WORKING ON ROUTING ....................................................................... 5
7. ROUTING REQUIREMENTS....................................................................................................... 6
8. ROUTING DESIGN SPECS......................................................................................................... 6
9. SUGGESTED APPROACH TO IMPLEMENT ROUTING ............................................................... 7
10. DEBUGGING YOUR CODE ...................................................................................................... 9
11. PROJECT ORGANIZATION .................................................................................................. 8
12. SUBMISSION REQUIREMENTS ............................................................................................ 9
13. SAMPLE DEDUCTIONS .......................................................................................................... 10
1. Overview

In this homework, you shall use JavaScript (Provider Style coding) to enhance your user interface two ways:

- Drop down menus.
- Do-It-Yourself routing – with a nav bar that just has two working links: home and blog.

You’ll also move part of your home content to blog content.

2. What is a Drop Down Menu?

A drop down menu is one where you click on a menu header and it opens up associated area that has detailed links. To facilitate communication, let me define two terms: Drop Down Header and Drop Down Content. The Drop Down Headers are what initially show in the menu while the Drop Down Content areas are initially hidden. When you click on a Drop Down Header, its associated Drop Down Content is toggled (hidden if showing, shown if hidden).

3. Before You Start Working on The Drop Down Menu

Back up your project before starting anything. You can merely copy your project folder somewhere outside of the NetBeans Project folder – give it a good name so you know what’s inside (e.g., homePageHW_final).

Check out the Tutorials – JavaScript page from my web site.

- This page introduces JavaScript as a language and tells you how to use W3Schools.com “Try-It-Yourself” JavaScript pages to learn more about JavaScript and answer any questions you may have.
- It also shows you how to use Chrome’s JS debugging tool which is really helpful.
- Carefully read JS File Naming Convention and JS SW Design sections. They are very important and this homework expects you to adhere to the SW design principles described therein.
- Study the Drop Down Menu Sample Code which includes the following:

  ✓ **CSS positioning** - review of fixed positioning, introducing absolute positioning that is used to right align each Drop Down Content area with its Drop Down Header.
  ✓ **JS Onclick Event** - adding an onclick attribute to HTML elements, calling a function that toggle (hides / shows) Drop Down Content when you click on the associated Drop Down Header. There is one example using words for Drop Down Headers and another that uses icons for Drop Down Headers. Icons work especially well in mobile format since they are smaller.
  ✓ **CSS transitions** to animate the toggling of Drop Down Content, e.g., fading in/out by changing opacity slowly or zooming in or out off the right of the screen.
  ✓ **JS code to add click events** adds a click event to every Drop Down Header (so that the “HTML coder” does not have to do that in the HTML code).
  ✓ **Window click code** demonstrates how you can hide all Drop Down Contents if the user clicks anywhere on the page (besides a Drop Down Header).
4. Drop Down Requirements and Design Specs

- When your home page first renders, its Drop Down Menu shall show the following (similar to the sample code): *Home, Account, Search, Tutorial, and Blog.*
  - The links in your nav bar may be **words** or **icons** that display those same words upon hover.
  - If you want to use icons, look at the icon sample code which supplies light and dark icons that you can use. You can also create/find your own icons – if you have time, but it will likely involve some basic image editing such as described from a link from Tutorials – Web Design.

- The Account, Search, and Tutorial entries in the menu shall be **Drop Down Headers**, whereas the Home and Blog will just be links (that do not hide/show anything). When the user clicks on a Drop Down Header, two things shall happen:
  - The associated Drop Down Content area shall toggle (show/hide) using a noticeable transition (e.g., fade in/out, slide in/out, or any other transition you want to create).
  - Any other Drop Down Content area currently open shall be hidden.

- Your nav bar shall have the following **Drop Down Content** areas:
  - The **Account** Drop Down Header shall be associated with a Drop Down Content with these links: *Register, Log In, Profile, and Log Out.*
  - The **Search** Drop Down Header shall be associated with a Drop Down Content with two links:
    - *Users* (or you could rename to something like Customers or Musicians or Travelers, depending on what your web site is about) and
    - *Something else* (whatever you decided to use for your “other” database table).
  - The **Tutorial** Drop Down Header shall be associated with a Drop Down Content with these links: *Proposal, Proof of Concept, Tutorial Home, and Demo.*
  - None of the links in your nav bar need to work for the Drop Down Portion of the homework (just the Drop Down Headers must open/close the Drop Down Content areas). However, when you get the routing part of this HW, your Home and Blog links WILL work.

- When the user clicks elsewhere on the page (e.g., clicking NOT on a Drop Down Header), all Drop Down Content areas shall be hidden.

- Your drop down code shall be in an **external JS file** using the JS file naming convention specified in Tutorials – JavaScript (file named the same as the SINGLE function or object defined within).

- Using the **Software Design techniques** described in Tutorials – JavaScript, your dropdown code shall:
  - apply the **Single Responsibility Principle** and
  - be written in **Provider style** (making things as easy as possible for the “HTML coder” while adhering to our **Simplified Dependency Injection** principle meaning that the code in the external JS file shall make NO ASSUMPTIONS about anything from the HTML page. Your HTML Page must pass in whatever the JS code needs to know about the HTML/CSS (see Tutorials – JavaScript, the SW design section).
  - minimize the number of globally defined elements.
• Fix any problems you may still have left over from your first homework. Even if the grades from the last HW are not completed yet, you should be able to assess your own page:
  o The titleNav shall be fixed to the top and the footer shall be fixed to the bottom (with appropriate padding to ensure no content is ever hidden by these two fixed elements).
  o The titleNav shall be responsive (title and nav sharing one line for desktop, title and nav each having their own line for mobile) and the top padding of the content area adjusting depending on the height of the titleNav.
  o The links in the nav shall be styled differently than the links elsewhere on the page (by using a compound CSS rule set).
  o The page shall have a fluid design, meaning that when you narrow the browser, the text re-wraps fitting into the narrower width (with no horizontal scroll bar). When you widen the browser, the text expands to fill out the available space.
  o At any browser width, no titles or links in the TitleNav shall wrap. Use “white-space: nowrap;”
  o All text (including links) shall be legible (large enough and with enough contrast to background).
  o No text shall be overly close to any visible edge.
  o The page shall not load slowly due to image files that are too large (check rendering the page after you have emptied your browser cache and/or check the file sizes of your images – keep the total size of all images on a page perhaps less than 750K).
  o Originality. Your layout shall be substantially different from sample code and your classmates.

5. Hints for Drop Down Solution

Here is how the sample code relates to your Drop Down Solution.

✓ CSS positioning – by styling each Drop Down Content to be absolutely positioned with respect to their container, we get a really nice look. All Drop Contents are always (at any viewport widths) placed relative to its container. The continer of Drop Down Content needs to be relative (the rule is that Absolute elements are placed with respect to their first non-static parent, so we make this work by assigning a relative position to the parent).

✓ JS Onclick Event – for your solution, you don’t need to add onclick attributes to your HTML code, but it is still valuable to study for the quiz.

✓ CSS transitions – CSS transitions certainly make your page look more professional. I provided two examples, one zooming in from right and the other fading in. You can also come up with your own transition.

✓ JS code to add click events - For your solution, you don’t need JS code to iterate through HTML elements adding onclick attributes, but it is still valuable to study for the quiz.

✓ Window click code – this demonstrates how you can add a window click even that will fire no matter where the user clicks on the page. Here’s some simple logic for you to apply: figure out if the user clicked on a Drop Down Header. If so, toggle the related Drop Down Contents (and close all other Drop Down Contents that may be open). If not, close all Drop Down Contents.

Note: if you get the drop down desired functionality to work but believe your software design is not optimal, go ahead and work on the routing functionality. The routing sample code goes a bit farther demonstrating the implementation of the SW design principles that have been prescribed for this homework. After completing the routing, come back and see if you can improve upon the design of your dropdown code.
6. Before You Start Working on Routing

Study the materials presented on my web page Tutorials – JavaScript (down to and including the section entitled “JavaScript Routing Code”. Below is a summary of the concepts presented in each code folder:

- **01_Two Pages Copy Paste** shows you how a beginner would create a web site – with no reuse of the User Interface code. They would just make a copy of the index page, call it blog.html, then modify the content area of the blog page. The links look like this: `<a href="blog.html">`. Whenever the user clicks on one of these links, they load a brand new page (even though only the content needs to change). The result looks pretty good, but the only problem is maintenance. If you had 100 pages in your site and then you needed to change the title or the footer or a nav link, you’d have to make that same change in 100 files. Not good!

- **02_Internal Links** just shows you how internal links work. When the href attribute of an `<a>` anchor tag looks like this `<a href="#myId">`, then clicking the link makes the page scroll down to whatever element has that id (on the page). No new page is loaded (which also means the JavaScript on the page is “still in control” of things).

- **03_Reuse without Routing** shows how to make JS components (that put some HTML into an element on the page). Since this version does not use any links -- the browser’s forward/back buttons cannot be used to go back to previous link.

- **04_Routing Multi-Branch If** uses links and introduces a routing function that is invoked whenever the browser’s address bar changes. Instead of using a sophisticated routing table, the router function has a multi-branch if statement that invokes the correct component function (e.g., home or blog) based on what link was clicked (e.g., #/home or #/blog).

- **06_Associative Array**. An associative array is like a “hash table” where you can store key/value pairs. With associative arrays, the “index” (inside the square brackets) is the key, usually a String, not an integer. The value can be anything, like a string or an object or a function.

- **10_Routing Table** uses a hash table (associative array) instead of a **multi-branch if** statement to determine which function to call based on the link in the address bar. The keys of the hash table are the links (for example, that the HTML coder has in their nav bar). The value for each link is a component function that puts HTML content into the content area. Here is an example of populating one element of the hash table to indicate that when the “#/home” link is in the browser’s address bar, the home (JS function) should be run.

```javascript
routes['#/home'] = home;
```

- **11_Parameter Objects** shows the benefits of using a parameter object instead of a parameter list. Parameter objects are more self documenting (you see property names to indicate what you are setting, instead of relying on order to let you know which parameter is which). It is also convenient for the HTML coder to skip setting any parameters s/he does not care about, allowing the provider code to just set those to default values.

- **13_Routing Framework** is encapsulated. It has consumer code (in the HTML page) and provider code (in the JS file). With this code, the consumer code (the JS in the HTML page) sets up the routing table and then passes this to the provider code (in the JS file) which handles everything from there.
7. Routing Requirements

- On first rendering, your home page shall show “home page” content in the content area. The URL in the browser’s address bar shall show something like \texttt{index.html/#home}. The Home page content shall include:
  - \textit{Your Advertised Web Site Functionality} — the 1-2 paragraphs that describe the functionality you propose for your website and entices users to visit your site.

- When the user clicks on the blog link (from your nav bar), your blog content shall show in the content area. The URL shall show something like \texttt{index.html/#blog}. Your Blog content shall include (each under a heading like <h2>):
  - \textit{Your Web Design Experience} — the paragraph that describes your web design/development experience.
  - \textit{Your Proposed Database Design} — the description of the database table you propose, that supports your advertised website functionality.
  - \textit{Your HomePage Homework Blog} — paragraph(s) that describe what you found easy, hard/confusing, and valuable about the first HW (Home Page).
  - \textit{Your JavaScript UI Blog} — paragraph(s) that describe what you found easy, hard/confusing, and valuable about this Homework.

8. Routing Design Specs

- Your web application shall reference an external JavaScript file that handles routing (let’s call the function in this file \texttt{routeFw}).

- \texttt{routeFw} shall take an input parameter object that specifies everything the routing code needs to know about the web page, like the routing table (that holds link/component function associations), the id of the content area, which component function should be displayed in the content area on first rendering.

- JavaScript code in the HTML page shall initialize the routing table and pass the necessary input parameter to the \texttt{routeFw} function.

- The code in the \texttt{routeFw} function shall add a listener for URL changes. When a URL changes, it shall see what the new URL is and invoke the corresponding component function (as specified by the “passed in” routing table).
9. Suggested Approach to Implement Routing

- Create two JavaScript components named `blog.js` and `home.js` and place them into folder `js/components` as was demonstrated by sample code “03_Reuse_without_routing”:

```
function home() {
    // ` this is a "back tick". Use it to define multi-line strings in JavaScript.
    var content = `<h4>Home</h4>
    <p>This is my Home Page Content !!! The home and blog links should work. </p>
    `;
    var ele = document.createElement("div");
    ele.innerHTML = content;
    return ele;
}
```

- Make the `home` function (in home.js) inject Your Advertised Web Site Functionality into the content page. This is the 1-2 paragraphs that describe your proposed web site functionality, that entices users to visit your site. You can model your code after function `home` in folder 03_Reuse_without_routing.

```
function home() {
    // ` this is a "back tick". Use it to define multi-line strings in JavaScript.
    var content = `<h4>Home</h4>
    <p>This is my Home Page Content !!! The home and blog links should work. </p>
    `;
    var ele = document.createElement("div");
    ele.innerHTML = content;
    return ele;
}
```

- Then create `blog.js` as a similar component, placing (in its content) these sections: My Web Design/Development Experience, My Database Design, HomePage Homework, JavaScript UI Homework (use headers like <h2> for each section).

- At this point, I recommend that you test your JS Components (and your understanding of getting this far in the sample code) by temporarily adding onclick events to the Home and Blog links in your nav bar like this:

```
<a onclick = "home('view')">Home</a> | 
<a onclick = "blog('view')">Blog</a>
```
• Compare your code to the 03_Reuse_without_routing and make sure:
  o Your index.html has script tags to reference each of your new JS components.
  o There is no HTML code in the content area of your index.html page. You get home content in the content area upon first rendering by initially invoking the home function.
  o Probably your home page homework has “content” as the id for your content area. The routing sample code used “view” for that purpose. Make sure you are injecting content into the element with id “content” and not “view” like the sample code does.
• After studying code examples 04 routing multi-branch if and 06 associative array, create a routing table as was demonstrated by sample code “10_Routing_table” and modify your code to work with a routing table. Test that you can link between home and blog and back.
• Test that if you are viewing home and you refresh the page, you still see content in the content area. If you do not, you don’t set a “dummy link” before setting the link that will invoke your initial home function. The sample code has this and should work properly.
  // invoke an invalid content/link before the initial link so there’s always a link change even when the user
  // refreshes the page when the link was already #/home. Link change is what kicks off the router function.
  window.location.hash = '#/xxx';
  window.location.hash = '#/home';
• After studying code example 11_parameter_objects, make your code reusable (framework style) as was demonstrated by sample code 13_Routing_FW. This code should employ all the JavaScript software design principles (Provider Style code that adheres to Single Responsibility Principle, simplified Dependency Injection, parameter object, follows the JS file naming convention).

10. Project Organization

Your project shall be organized as shown below.

• In the js folder, differentiate between general purpose, reusable JS code (e.g., dropdownFw.js and routeFw.js) and specific “components” (e.g., home.js and blog.js) that just define how the content area should look.
11. Debugging Your Code

Before viewing your page, look for red syntax error bubbles in NetBeans and address any problems. Place console.log statements anywhere in your code so that you can follow what’s happening when the page runs. Once you view your page, make sure to have clicked F12 so that you’ll see your console.log statements as well as any possible JS runtime errors. Try out Chrome’s debugger (there’s a section on that with video from Tutorials – JavaScript).

12. Submission Requirements

When you have completed all the requirements, tested locally (syntax checked), and published (and tested what you published), submit a zip file of your web site into Canvas (No RARs). Your zip file submission is important because we base the late penalty off of the date/time of submission. Also, if there is ever a question about how we graded a homework, we will go by the code that was submitted, not by what is currently published.

Into the text area of your Canvas submission, perform a self assessment. This consists of you grading your own homework, using the list of deductions from the end of this document. Enter one line per deduction (numeric deduction first, followed by deduction explanation). At the top, type in the total grade you expect to receive for the HW. If you wish, you and a friend/classmate can grade each other’s HW. And, of course, you can fix whatever’s broken and remove the related deduction as long as it’s before the due date.
13. Sample Deductions

Submission

- **9 for Not Publishing**: We perform functional testing each week based on what you have published. You need to test locally, publish, then test what you published.
- **9 for Lack of Canvas Submission**: If there is ever any question about a grade, we go by the code that was uploaded into Canvas at the time the assignment was due.

Drop Down

- **-5 if Drop Down Menus not implemented.**
- **-2 if clicking a Drop Down Header shows but doesn’t hide** the associated Drop Content area.
- **-1 if clicking one Drop Down Header does not hide any shown Drop Down Content area.**
- **-1 if clicking elsewhere (besides on a Drop Down Header) does not close all Drop Down Content areas.**
- **Up to -1 for missing entries in nav bar or if icons used without hover words.**
- **-0.5 if no CSS transition is used to toggle Drop Down Content areas.**

Routing

- **-5: if routing/reuse not implemented** (maybe the student copied index.html to blog.html and just linked between the two).
- **-3: if reuse implemented but without routing table** (maybe the student used an approach like 03_reuse_without_routing).
- **-0.5: using a parameter list (passing into routeFw) instead of a parameter object.**
- **-1: page not always initialized properly.** if you view the home page then refresh the page and the content shows up empty (it’s missing setting window.location to a dummy value before setting it to the home function therefore there’s no link change event invoking the routing function.
- **Up to -2: project files not organized per spec (see diagram in this HW writeup).**

Blog

- **Up to -1.5 for missing/insufficient Blog Entry for this HW.**
- **Up to -1 for missing/insufficient Blog Entry from previous HW.**

General

- **Up to -2 for Unprofessional / Inappropriate Home Page Content**: The “marketing material” from your home content area should be of a quality that acceptable by a “real company” that might be paying you to create their web site. The marketing material (what you say your web site has to offer)
must be able to be supported by your choice of layout for your “other” database table. The “blog” text from previous homeworks shall be removed from the home content (and moved to the blog content).

- **Up to -2 for Look and Feel:** Make sure you are still meeting the layout requirements for the first homework (Home Page). If you had issues, you should have fixed them by now. For example, you should have a reasonable layout that does not render slowly (e.g., due to too large image files) and that responds well when you narrow and expand the width of the browser (with no unwanted text wrapping or overlapping). All text should be legible (light on dark or dark on light, no busy background image that makes it hard to read the text). There should be no text close to any visible edges. Footer should be fixed to the bottom and never cover up any content. TitleNav should be fixed to top (and never cover up any content). TitleNav should be responsive (meaning title and nav share a line in desktop but each have their own line in mobile, content top padding should adjust depending on the height of the titleNav). There should be substantial differences between your layout and the sample code and your classmate’s layouts.

- **Up to 2 if Syntax Errors** when we View Source (in Firefox) and we see red syntax error messages.

**Self Assessment**

- **Up to -2 for missing/insufficient self assessment.**