**Assignment 7 – Javazon**

**Setup**

Each week you will be asked to create a new folder under web-231 following a naming convention of “week-<number>.” If we are on week two, the folder name should be “week-2.” All files associated with the weekly assignment will be added to the appropriate weekly folder. All programs must be linked in the index.html landing page under the “Weekly Assignments” section. Projects will be linked under the “Projects” section of the index.html landing page. To be clear, **all** of the JavaScript, HTML, images, and CSS files associated with a weekly assignment must be placed under the appropriate weekly folder. The page title for all HTML files in this course must say “WEB 231 – Enterprise JavaScript I.” And, all HTML and CSS files must be valid HTML/CSS, tested through the WC3 validator. The links were provided during WEB 200 and were added to the index.html landing page. Also, the blue border around the provided images is to show they are images and should not be included in your submission. In other words, do not add a blue border around your work, unless the instructions explicitly ask for it.

**User interface styling and formatting requirements are located in the HTML, CSS, and JavaScript Requirements document.**

HTML: **<yourLastName>-javazon.html**

CSS: **<yourLastName>-javazon.css**

**Grading Reminders**

1. (50%-points) All code sources (.html, .css, .js) must be cited in the opening programmers’ comments, following the format specified in the code attribution document.
2. (25%-points) All code sources (.html, .css, .js) must show evidence of code comments. This means each section of the program (.html, .css, .js) must include code comments that explains what the block of codes purpose is, what the required parameters are (data type, if any), and what the expected output is.
3. (rubric) All code sources (.html, .css, .js) are measured against
   1. Code functionality: Does it work? Does it meet requirements?
   2. Adherence to standards and conventions. Are you using the appropriate data types, including proper indention, are variables named appropriate (variable x is an example of poor naming conventions), is there an appropriate use of whitespace, is the code organized, and are semicolons being used to terminate code sentences?
   3. Efficiency: Use of language features. Are you practicing DRY (Don’t-Repeat-Yourself?), are you leveraging built-in language features where appropriate, and are you using classes/functions to reduce code clutter?
   4. Documentation: Self-documenting, naming conventions, code is maintainable by others. Is the code your write easy to read and maintainable by others?
   5. Error trapping/handling. Are there errors in the program? Is there evidence of coding best practices to reduce user errors?
   6. Assignment Specific Compliance. Does the delivered solution follow the instructions, as they are written? Does the output match what was provided in the screenshots (including spaces, styling, etc.)?

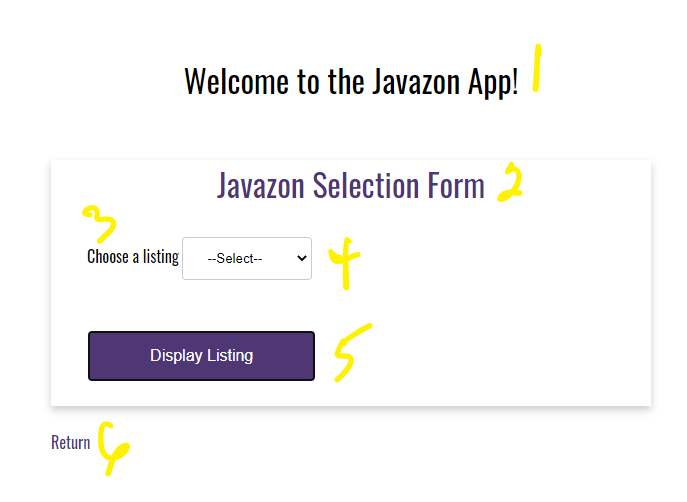
**Required Modifications**

* Cite any sources in your opening programmer’s comment
* Link the appropriate CSS files and Google fonts
* onload…

**Additional JavaScript Requirements**

1. Create a Product class with a constructor with 4 parameters: id, description, price, and quantity. In the body of the constructor add a fifth property called **totalValue**. To calculate the total value, take the quantity and multiply it by the price (Exhibit B, #5). Set the decimal placement to two (Exhibit B, #5).
2. Create a Services class with a constructor with 4 parameters: id, description, hourlyRate, and min. Set the **hourlyRate** decimal placement to two.
3. Create an Employee class with a constructor with 4 parameters: id, firstName, lastName, salary. Set the salary decimal placement to two.
4. Create 5 new Product objects and assign them to 5 separate variables.
5. Create 3 new Services objects and assign them to 3 separate variables.
6. Create 5 new Employee objects and assign them to 5 separate variables.
7. Using document.getElementById, assign the products div (Exhibit B) to a variable named products.
8. Using document.getElemetnById, assign the services div (Exhibit C) to a variable named services.
9. Using document.getElementById, assign the employees div (Exhibit D) to a variable named employees.
10. Set the products div display to none (Exhibit A).
11. Set the services div display to none (Exhibit A).
12. Set the employees div display to none (Exhibit A).

**Exhibit A. User Interface (Initial view)**

****

1. h1 with a CSS class of app-header and a text value of “Welcome to the Javazon App!”
2. card-title with a text value of “Javazon Selection Form”
3. form-field label with a text value of “Choose a listing.”
4. HTML select list with a CSS class of drop-down-menu and an id of listing.

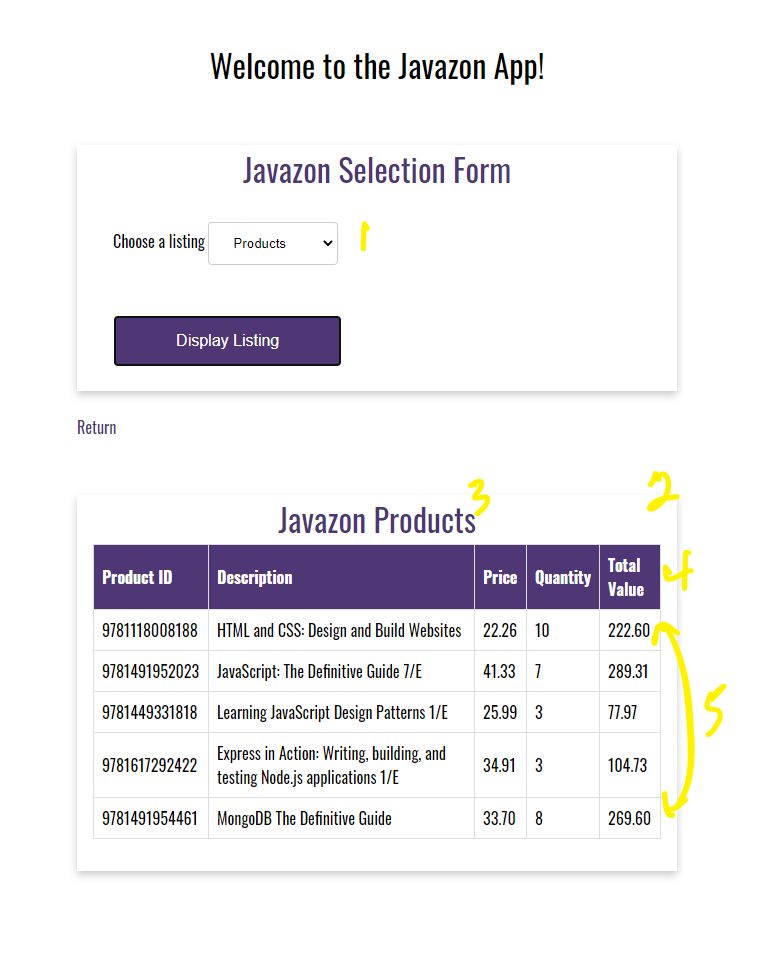
**Additional HTML Requirements**

1. The select form-field will need 4 HTML options
2. value=select; text=--Select--
3. value=products; text=Products
4. value=services; text=Services
5. value=employees; text=Employees
6. HTML button with an id of btnDisplayListing and a text value of “Display Listing”

**Additional JavaScript Requirements**

1. Register an onclick event for btnDisplayListing.
2. Capture the value from the select list and assign it to a variable named **choice**.
3. Bind the properties of your 5 Product objects to an HTML table.
4. Bind the properties of your 3 Services objects to an HTML table.
5. Bind the properties of your 5 Employee objects to an HTML table.
6. Using a JavaScript switch statement, use **choice** for the switch expression and a series of case statements to compare **choice** against the string values: products, services, and employees.
7. For products (Exhibit B, #1), set the products div display to block, set the services div display to none, and set the employees div display to none.

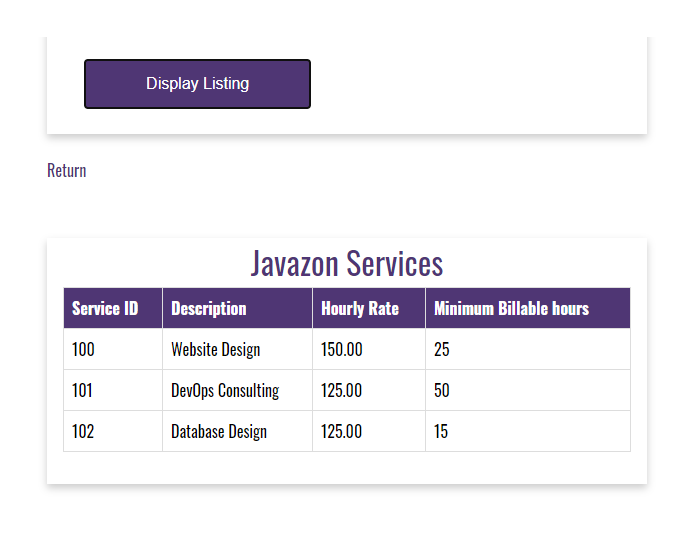
**Exhibit B. Products**

****

**Additional Programming Assistance**

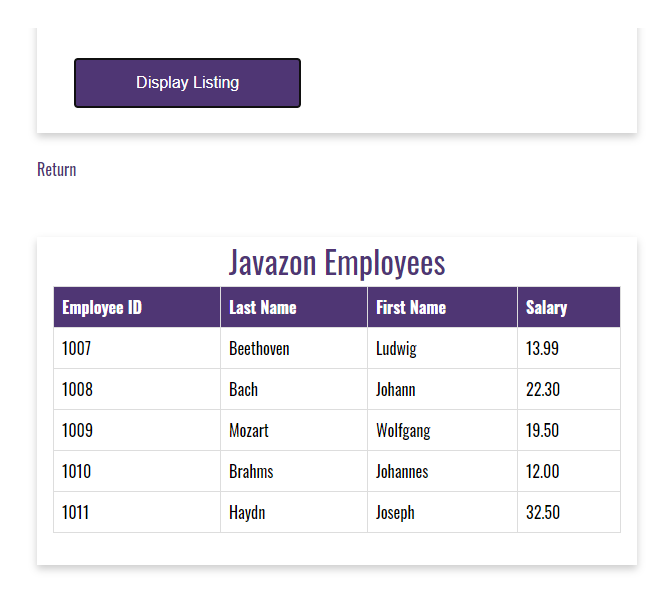
1. User selection.
2. products div; this is using a card with a card-title and card-content.
3. card-title with a text value of “Javazon Products.”
4. Table header row. Notice the styling. Does this look familiar? You guessed it, reuse the table styling from Assignment 6.
5. Notice the decimal placement for **Price** and **Total Value.**
6. The “Total Value” data is coming from the Product classes **totalValue** field, which is calculated by multiplying the properties quantity and price.
7. For services (Exhibit C), set the services div display to block, set the products div display to none, and set the employees div display to none.

**Exhibit C. Services**

****

1. For employees (Exhibit D), set the employees div style to block, set the products div display to none, and set the services div display to none.

**Exhibit D. Employees**

****

1. For default (Exhibit A), set the products div display to none, set the services div display to none, and set the employees div display to none.

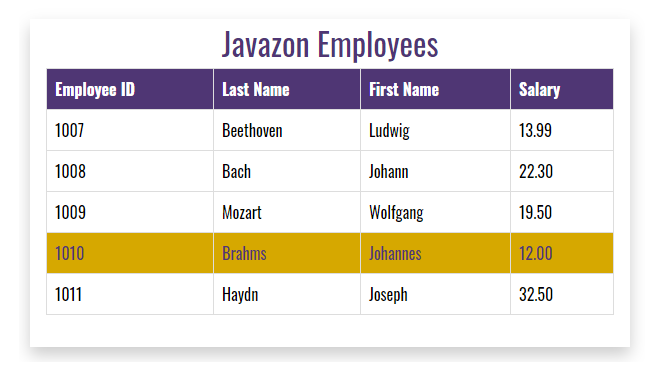
**Additional Assistance**

1. Exhibit B, C, and D show the anchor link between the displayable tables. This means the anchor link should be placed in the “Javazon Selection Form.”
2. In order to bind the class properties to each of the appropriate tables, you will need to give the table data elements unique ids.
3. Pay close attention to the decimal placement in each of the tables.

**Additional Styling Requirements**

1. All three HTML tables will follow the styling we used in Assignment 6. There are several ways to do this, but the easiest way is to simply copy the table styling from Assignment 6 and paste it into this week’s CSS file.
2. Each table will be placed in its own card, with the correct title. This means you will have a total of three div sections: products, services, and employees.

**Exhibit E. Table Styling**

****

1. anchor link with the CSS class return-home and a link back to the index.html landing page.