

## Assignment 3 – Restaurant App

### Setup

Each week you will be asked to create a new folder under web-330 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 folder. All programs must be linked in the appropriate landing page. Projects will be linked under the “Projects” section of the index.html landing page.

The document title of all HTML files in this course must say “WEB 330 – Enterprise JavaScript II.” And, all HTML and CSS files must be valid HTML/CSS, tested through the WC3 validator. The links were provided during WEB 200 and 231. As part of your submission, be sure to include screenshots of the results from the validation tests (HTML and CSS validators).

**User interface styling and formatting requirements are located in the Web 330 HTML, CSS, and JavaScript Requirements document.**

HTML: **restaurant.html**

CSS: **restaurant.css**

JS: **product.js, appetizer.js, beverage.js, bill.js, dessert.js, main-course.js, index.js**

### Grading Reminders

- a. (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: Code is maintainable by others
  - i. Code comments are present in all blocks of code, written as full sentences, free of grammatical errors, and function/class parameters and data types have been identified.
  - ii. Code attribution is present in all files and authorship is clearly annotated.
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, JavaScript, and Google fonts
- product.js

### Additional JavaScript Requirements

- a. Create a JavaScript class named Product. Give the class a constructor with two parameters: name and price. Export the class. See Exhibit A.

#### Exhibit A. Product class

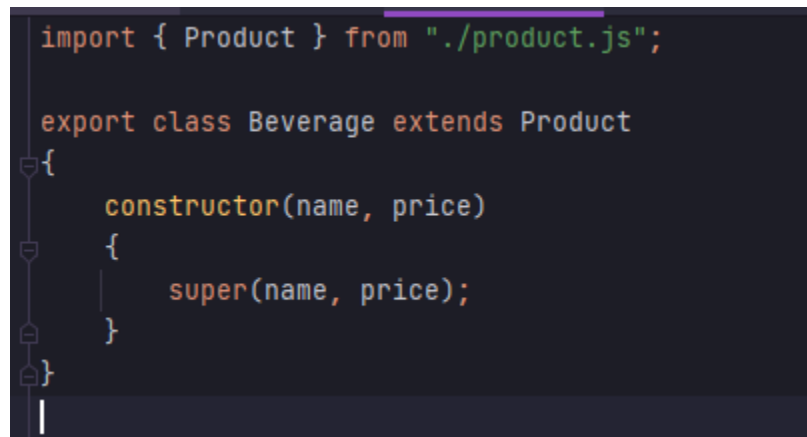
```
export class Product
{
  constructor(name, price)
  {
    this.name = name;
    this.price = price;
  }
}
```

- berverage.js

### Additional JavaScript Requirements

- Create a JavaScript class named Beverage. Add an import statement for the product.js file and inherit the base classes properties by referencing the extends keyword. In the class's constructor add two parameters: name and price. Pass these parameters to the parent through the super() function. See Exhibit B.
- Export the class (see Exhibit B).

### Exhibit B. Class Inheritance



```
import { Product } from './product.js';

export class Beverage extends Product
{
  constructor(name, price)
  {
    super(name, price);
  }
}
```

- appetizer.js

### Additional JavaScript Requirements

- Add an import statement for the Product object.
- Create a class named Appetizer and inherit the Product object.
- In the constructor of the class add two parameters (name and price) and pass them to the parent class through the super() function.
- Export the class.

- main-course.js

### Additional JavaScript Requirements

- a. Add an import statement for the Product object.
  - b. Create a class named MainCourse and inherit the Product object.
  - c. In the classes constructor add two parameters (name and price) and pass them to the parent class through the super() function.
  - d. Export the class.
- dessert.js

**Additional JavaScript Requirements**

- a. Add an import statement for the Product object.
  - b. Create a class named MainCourse and inherit the Product object.
  - c. In the classes constructor add two parameters (name and price) and pass them to the parent class through the super() function.
  - d. Export the class.
- bill.js

**Additional JavaScript Requirements**

- a. Create a class named Bill.
- b. Create four class properties: \_beverages, \_appetizers, \_mainCourses, and \_desserts and assign them an empty array (opening and closing brackets).
- c. Create a function named addBeverage() with one parameter of type beverage. In the body of the function push the beverage object to the \_beverages array. See Exhibit C.

**Exhibit C. addBeverage() function**

```
addBeverage(beverage)
{
  this._beverages.push(beverage);
}
```

- d. Create a function named `addAppetizer()` with one parameter of type `appetizer`. In the body of the function push the `appetizer` object to the `_appetizers` array. Refer back to Exhibit C. for an example of what I am looking for.
- e. Create a function named `addMainCourse()` with one parameter of type `mainCourse`. In the body of the function push the `mainCourse` object to the `_mainCourses` array. Refer back to Exhibit C. for an example of what I am looking for.
- f. Create a function named `addDessert()` with one parameter of type `dessert`. In the body of the function push the `dessert` object to the `_desserts` array. Refer back to Exhibit C. for an example of what I am looking for.
- g. Create a function named `getTotal()`. In the body of the function create a variable named `total` and assign it a default value of 0. Next, using JavaScript's built-in `forEach()` function, loop over each array (`_beverages`, `_mainCourses`, etc.,) and calculate the total using the `price` field of each object (see Exhibit D). In the `forEach()` body, be sure to use JavaScript's `parseFloat()` function to ensure the calculated results are a float. Finally, return the `total` variable and set its precisions to two decimal places.
- h. Export the class.

#### Exhibit D. JavaScript's built-in `forEach()`

```
let total = 0;

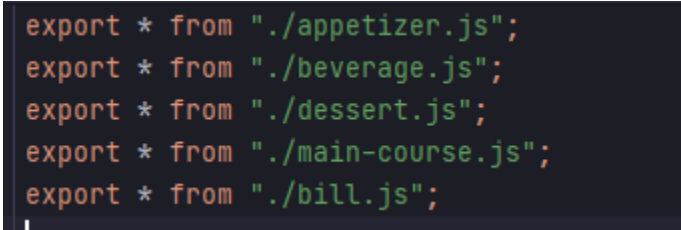
let beverageTotal = this._beverages.forEach(function(beverage)
{
    total += parseFloat(beverage.price);
})
```

- `index.js`

#### Additional JavaScript Requirements

- a. Add export statements for each of the classes you created in the `index.js` file. See Exhibit E. As a side note, we are doing this to make it easier for the imports in the `restaurant.html` file. One of the tricky parts about modules in JavaScript is how you are allowed to import duplicate files. For example, in the `MainCourse` class we are importing

the Product class. If we try to import MainCourse directly from the restaurant.html file we will likely run into an error, because JavaScript doesn't like importing files that have nested imports. This is where the index.js file comes into play. If export all of the files from single file JavaScript thinks all imports are only one layer deep.

**Exhibit E. index.js file**

```
export * from "./appetizer.js";  
export * from "./beverage.js";  
export * from "./dessert.js";  
export * from "./main-course.js";  
export * from "./bill.js";
```

**Exhibit F. User Interface (final solution)**

Welcome to the Restaurant App! 1

### Restaurant Menu 2

#### Beverages 3

- ☒ Soda (\$3.30)
- ☐ Sweet Tea (\$3.30) 4
- ☐ Iced Coffee (\$2.49)
- ☐ Bottled Water (\$2.15)

#### Appetizers 5

- ☒ Nachos (\$6.60)
- ☐ Chips and Salsa (\$5.50) 6
- ☐ Pretzel with Cheese (\$8.80)
- ☐ Fried Pickles (\$8.25)

#### Main Courses 7

- ☐ Street Tacos (\$9.70)
- ☒ Boneless Wings (\$12.80) 8
- ☐ Cheese Burger with Fries (\$12.65)
- ☐ Chicken Wrap with Salad (\$9.65)

#### Desserts 9

- ☐ Ice Cream (\$4.30)
- ☐ Chocolate Cake (\$5.40) 10
- ☒ Apple Pie (\$3.80)

Place Order 11

Return 12

### Order Summary 13

Your order total is 26.50 14

1. h1 with the CSS class app-header and a text value of “Welcome to the Restaurant App!”
2. card-title with a text value of “Restaurant Menu”
3. card-title with a text value of “Beverages”
4. card-content with four labels and four checkbox input fields
5. card-title with a text value of “Appetizers”
6. card-content with four labels and four checkbox input fields

7. card-title with a text value of “Main Courses”
8. card-content with four labels and four checkbox input fields
9. card-title with a text value of “Desserts”
10. card-content with four labels and four checkbox input fields
11. button with an id of btnOrder and a text value of “Place Order”
12. anchor tag with a text value of “Return” and a link back to the index.html landing page
13. card title with a text value of “Order Summary”
14. card-content with an id of order-total and a CSS class of assign-results-text

### <yourLastName>-restaurant.html

- a. Set the script tag to type module.
- b. Import the Bill, Beverage, Appetizer, MainCourse, and Dessert classes from the index.js file.  
See Exhibit G.

#### Exhibit G. import statements

```
<script type="module">
  import { Bill, Beverage, Appetizer, MainCourse, Dessert } from "./index.js";
```

- c. Register an onclick event for the btnOrder element

#### Additional JavaScript Requirements (body of the onclick event)

- 1) Create variables to hold the divs where the checkboxes are placed in the HTML document. See Exhibit H and I.

#### Exhibit H. HTML div section



```
<div class="card">
  <div class="card-title">Main Courses</div>
  <div class="card-content" id="main-courses">
    <label for="tacos">
      <input type="checkbox" id="tacos" name=
    </label><br />
    <label for="wings">
      <input type="checkbox" id="wings" name=
    </label><br />
    <label for="burger">
      <input type="checkbox" id="burger" nam
    </label><br />
    <label for="wrap">
      <input type="checkbox" id="wrap" name=
    </label><br />
  </div>
</div>
```

**Exhibit I. JavaScript variables for the main-course div and checkboxes**

```
const mainCoursesEl = document.getElementById("main-courses");
const mainCourses = mainCoursesEl.getElementsByTagName("input");
```

- 2) **\*\*WARNING\*\*** Exhibit H. and Exhibit I. only demonstrate how to build one of the div sections and how to assign the section to JavaScript variables. You will need to apply this logic to the other three sections: Beverages, Appetizers, and Desserts.
- 3) Underneath the variables created in item 4, create a new instance of the Bill class. If you are not sure how to create a new class instance, refer back to the Reading & Videos section.
- 4) Using a for...of loop, iterate over the variables that contains the "input" fields. In the body of the for loop and an if statement that checks if the looped over value is checked. If the item is checked, call the appropriate add function from the Bill class and pass-in the name and value. See Exhibit J.

**Exhibit J. for...in loop over the mainCourses variable**

```
for (let mainCourse of mainCourses)
{
    if (mainCourse.checked)
    {
        bill.addMainCourse(new MainCourse(mainCourse.name, mainCourse.value));
    }
}
```

- 5) **\*\*WARNING\*\*** Exhibit J. only demonstrate how to iterate over one of the div sections. You will need to apply this logic to the remaining three sections: Beverages, Appetizers, and Desserts. The functions you created in the Bill class should support adding each section to the appropriate array.
- 6) Underneath the code for item 4-5, find the “order-total” div and bind the results from the getTotal() function. See Exhibit K.

**Exhibit K. order-total div**

A screenshot of a web application showing an order summary. The text "Order Summary" is displayed in a large, bold, blue font. Below it, the text "Your order total is 26.50" is displayed in a large, bold, black font. The background is a light gray gradient.

Order Summary

Your order total is 26.50