Case studies

The case studies are designed to give you practice developing substantial JavaScript and jQuery applications using the skills that you learned in *Murach's JavaScript and jQuery (4th Edition)*. The first case study has you develop an application that accepts an order from the user. You can develop this application after you read chapters 1 through 8.

The second and third case studies have you develop an application for a memory game. You can develop the first version of this application after you read chapters 1 through 15. After you read chapters 16 and 17, you can create a more professional version of this application that uses objects and libraries.

General guidelines

Starting files

- In most cases, your instructor will give you the starting files for these case studies, which include the HTML, CSS, and image files. Then, you will need to provide the JavaScript and jQuery that make the applications work.
- If one of the requirements for your course is a knowledge of HTML and CSS, you may be asked to create the applications from scratch. In that case, you will be provided with only the images for the applications.

User interfaces

- You should think of the user interfaces that are shown for the case studies as starting points. If you can improve on them, especially to make them more user-friendly, by all means do so.

Specifications

- You should think of the specifications that are given for the case studies as starting points. If you have the time to enhance the applications by improving on the starting specifications, by all means do so.

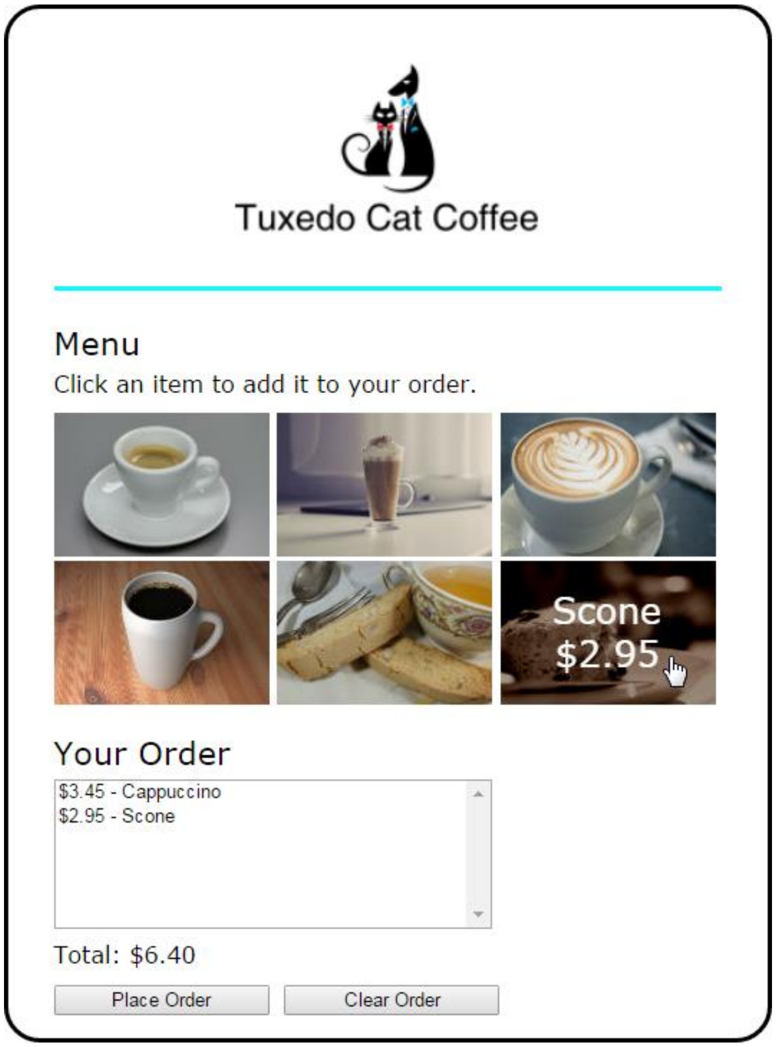
Top-down development

- Always start by developing a working version of the application for a case study. That way, you'll have something to show for your efforts if you run out of time. Then, you can build out that starting version of the application until it satisfies all of the specifications.

Case study 1: Create the Order application

For this case study, you'll create an application that allows the user to place an order at a fictitious coffee shop named Tuxedo Cat Coffee. Prerequisites: Chapters 1 to 8.

User interface



$3.45 - Cappuccino
$2.95 - Scone

Total: $6.40

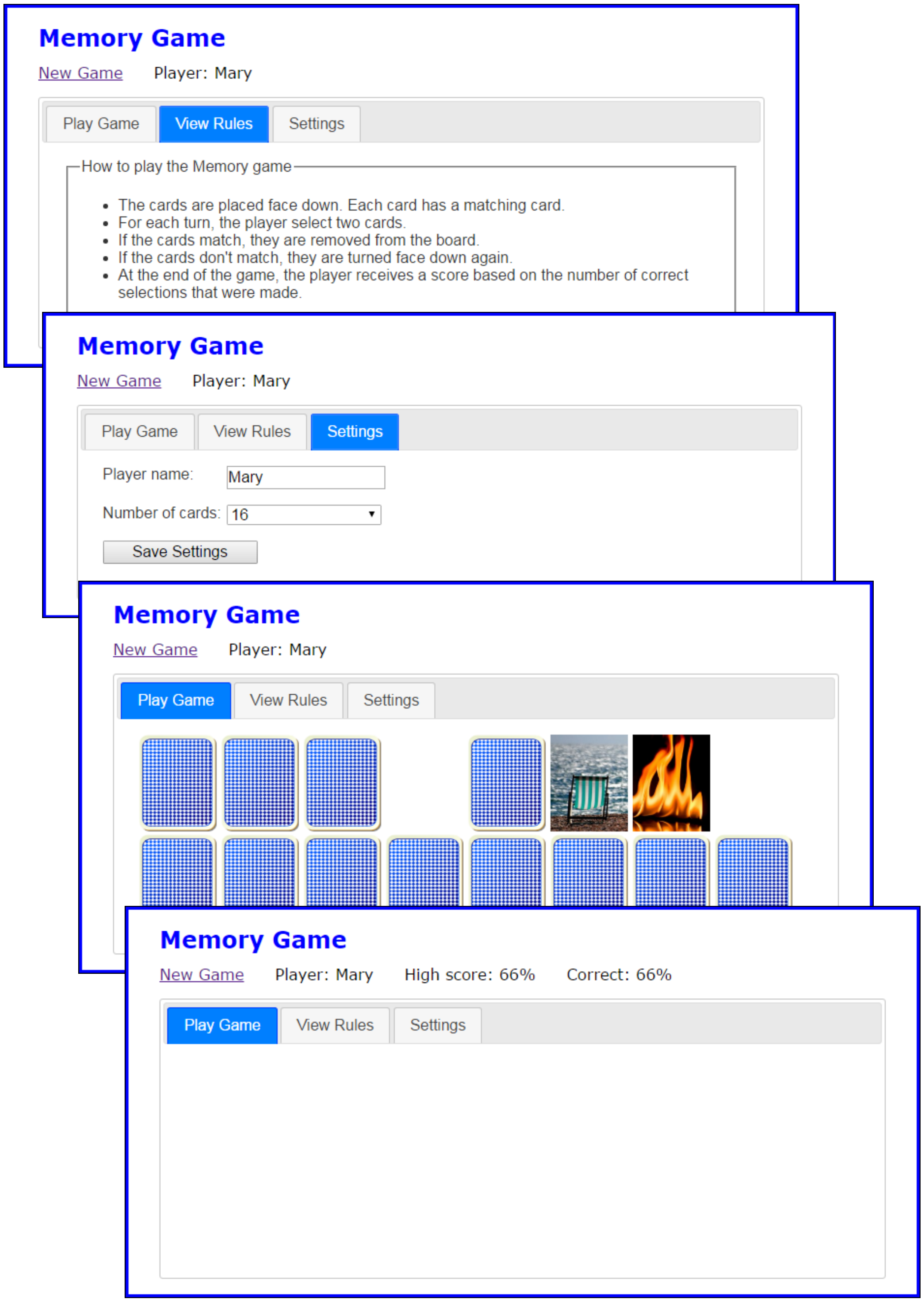
Specifications

- When the user hovers the mouse over one of the images in the menu, another image should be displayed with the description and price of the item. The id attribute of each image identifies the image to be displayed when it's rolled over.
- The rollover images should be preloaded.
- When the user clicks on an image, the order list and order total should be updated and displayed.
- If the user clicks the Place Order button, the checkout.html page should be displayed.
- If the user clicks the Clear Order button, all of the items should be removed from the order list and the total should be cleared.

Case study 2: Create the Memory game

For this case study, you'll create an application that allows the user to play the Memory game. The objective of this game is to turn over two cards at a time, trying to find cards that match until all cards have been matched. Prerequisites: Chapters 1 to 15.

User interface



Memory Game

New Game Player: Mary

Play Game View Rules Settings

How to play the Memory game

- The cards are placed face down. Each card has a matching card.
- For each turn, the player select two cards.
- If the cards match, they are removed from the board.
- If the cards don't match, they are turned face down again.
- At the end of the game, the player receives a score based on the number of correct selections that were made.

Memory Game

New Game Player: Mary

Play Game View Rules Settings

Player name: Mary

Number of cards: 16

Save Settings

Memory Game

New Game Player: Mary

Play Game View Rules Settings

Memory Game

New Game Player: Mary High score: 66% Correct: 66%

Play Game View Rules Settings

Specifications

- The main portion of the user interface should be displayed in a Tabs widget with the three tabs shown above.
- Images are provided for 24 cards, the back of the cards, and a blank card. All these images should be preloaded when the application starts.
- By default, the game should display six rows of cards with eight cards in each row, for a total of 48 cards (two of each card). The user can change the number of cards used as well as the player name from the Settings tab.
- When the user clicks the Save Settings button, the player name and number of cards should be saved in session storage. In addition, the page should be reloaded so the player's name, the player's high score if previous games have been played, and the correct number of cards are displayed.
- Use an images array to store the images for the cards used by a game. Use a cards array to store the src attributes of the images for the cards that will be displayed on the board (two for each image).
- Create the HTML for the displayed cards by randomly selecting elements from the cards array. The cards for each row should be stored in a div element, and the div elements for all the rows should be stored in the div element with an id of "cards".
- When the user clicks on a card whose back is displayed, the back of the card should be faded out over half a second and the front of the card should be faded in over half a second.
- When the user finds two matching cards, the cards should be hidden after one second using a sliding motion over half a second. If the cards don't match, they should be faded out over half a second after two seconds and the back of the cards should be faded in over half a second.
- Each time the user completes a game, the user's high score should be updated and displayed and the percentage of correct selections for the game that was just completed should be calculated and displayed. The high score should also be stored in local storage so it can be compared against the score for the user's next game.
- When the game ends, the user can click the New Game link to start another game.

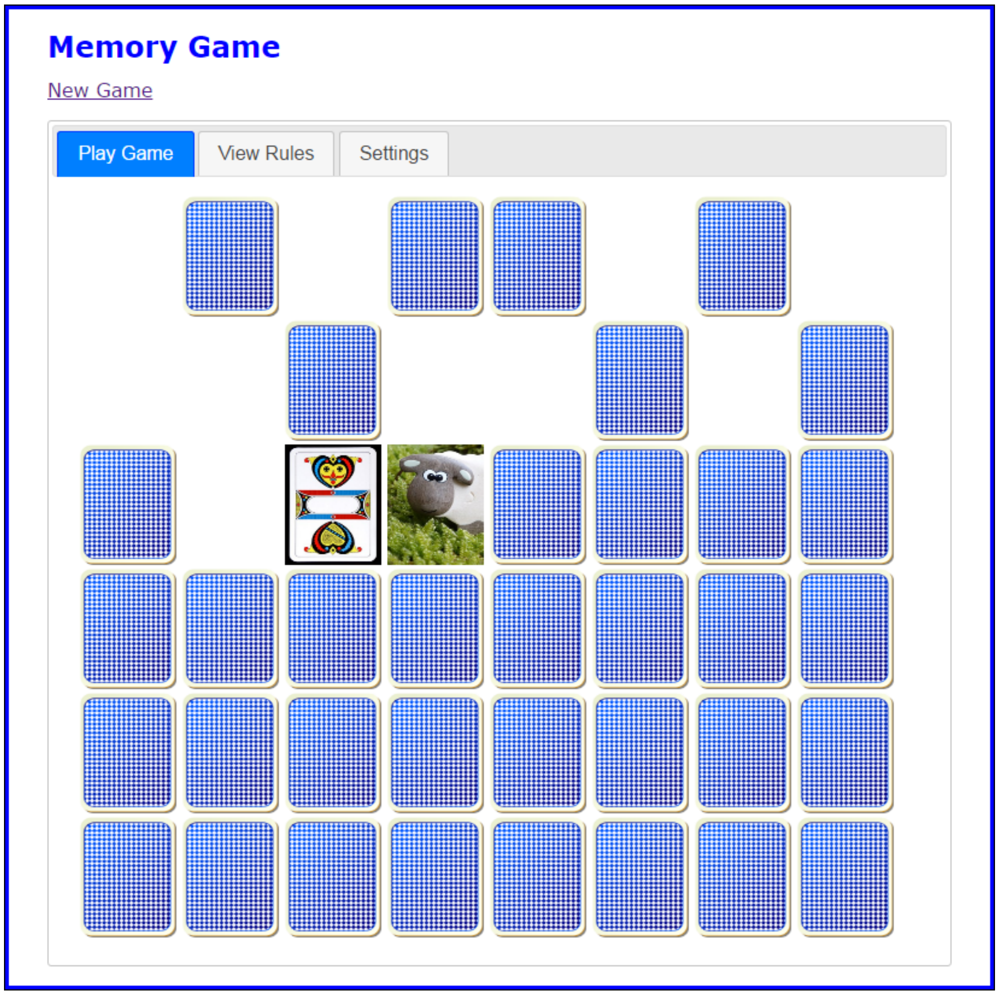
The HTML for each card

- The HTML for each card should consist of an <a> element with its id set to the src attribute for the card and its href attribute set to "#". The <a> element should contain an img element with its src attribute set to the source of the image for the back of the card and its alt attribute set to "".

Case study 3: Modify the Memory game to use objects

For this case study, you'll modify the Memory game from case study 2 so it uses objects and libraries. If you haven't already done case study 2, you'll need to do it first unless it's given to you as a starting point. Prerequisites: Chapters 1 to 17. *Note: Because the user interface is the same as in case study 2, the default interface is shown below.*

User interface



Specifications

- Rework the code for the application so it uses objects. Note that this will entail turning some existing functions into methods of an object, turning some existing variables into properties of an object, and turning some existing variables and functions into private state. Also note that as you make these changes, you'll sometimes need to adjust the existing code or add new code.
- The objects should be coded in these four JavaScript library files: library_settings.js, library_scores.js, library_cards.js, and library_card.js. These files are in addition to the main.js file.
- The library_settings.js file should create a settings object that keeps track of the player name and the number of images used for a game. This object should contain accessor properties that save this data to and retrieve it from session storage. Because you need just a single instance of the settings object and you don't need to protect any private state, you can create the settings object with an object literal.

- The library_scores.js file should create a scores object that keeps track of the score for each game as well as each player's high score. This object should contain the two integer variables that keep track of the number of turns and the number of turns that result in matches. It should also contain the functions that save the scores to and retrieve the scores from local storage. Because the integer variables and the functions that work with local storage should be in private state, you should use an IIFE and the module pattern to create the scores object.

  The IIFE should have public methods that increment the integer variables, check if all the cards have been removed from the board, calculate the percentage of correct selections, compare scores, and display the high score. The public methods that increment the integer variables should replace code that previously worked directly with those variables.

- The library_cards.js file should create a cards object that works with the cards. This object should contain the functions that preload and store images, store the src attributes for the cards, create the HTML for the cards, flip a card using a fade effect, and flip a card using a slide effect. Because the first two of these functions should be in private state, you should use an IIFE and the module pattern to create the cards object.

  In addition, the application should be modified so instead of using strings throughout the application to specify the src attributes for the card back and blank card, these attributes are stored in variables in this library. These variables should be in private state so they're protected, but outside code should be able to access them via read-only properties. A read-only property should also be used to get the number of images that's stored in a variable in private state by the function in private state that creates the card images.

- The library_card.js file should create a Card object for working with a single card when it's clicked. Since a new Card object will need to be created each time a card is clicked, a class should be used to define a Card object. Its constructor should accept a parameter that represents the <a> tag that was clicked, and it should use the jQuery selector to get the img tag that's a child of the <a> tag. The Card object type should have two properties that store this img object and the value in the id attribute of the <a> tag. The Card object type should also include two methods. The first one should check whether the user has clicked on a card that is blank or has already been revealed. The second one should accept a parameter that represents the first card that was clicked, and it should check if the id attribute of the <a> tag for this card is equal to the id attribute of the <a> tag for the current card.

- The library files should be included in the index.html file in the proper order, so each library is included after any libraries it depends on.