Competency 140 – JavaScript Objects and State

Module Plan

Leader: L Shumlich
Date: 2019/02

Overview & Purpose

Exercises to practice JavaScript classes, objects, prototypes, and React. In JavaScript it's all a little confusing. The intent of these exercises is to clear that up.

Prerequisite

- ReactJS
- JavaScript TDD
- Udemy - The Complete Web Developer - Zero to Mastery - To Section 19

Exercise - Plain Old JavaScript Class - Account

Definition of State according to Google:

"The particular condition that someone or something is in at a specific time"

For the rest of this assignment let's not think about "State" according to React but State as it relates to Computer Science. In software, state is everything. The current values of a thing is considered the thing's state. In most cases the thing is an object. What values the object have is considered to be the object's state.

Consider an 'Account'. A bank account or investment account. To make life simple there are no interest or bank charges.

Consider a class called 'Account' with the following methods:

- constructor - 2 parms - initial balance, account name
- deposit - 1 parm - value to be added
- withdraw -1 parm - value to be withdrawn
- balance - 0 parms - return current balance

In this case, state is only one thing "balance". Each method does something to the state of the object. It modifies it or shows it.

Methods are verbs or action words. State is typically nouns. Consider how we would test such an Account. Don't do this; just consider the following:

1. Create an account 'checkingAccount' with an initial balance of $25
2. Check the balance to ensure we have $25
3. Deposit $10
4. Check the balance to ensure we have $35
5. Withdraw $30
6. Check the balance to ensure we have $5

Let's consider the terminology of this example:

- What is the class? Account
- What is one of the objects or instances? checkingAccount
- What are the methods? constructor, deposit, withdraw, balance
- What are the parameters? to the constructor (startingBalance, accountName), to deposit (amount), to withdraw (amount), to balance() nothing.
- What is returned from? the constructor - reference to the new object, deposit - nothing, withdraw - nothing, balance - the balance.

Enough reading and thinking….not enough coding. Work in groups but code individually. Remember to keep your repository updated often.

Let's start:

- using TDD what order are we going to develop this in? Discuss in the group.
- create your JavaScripts in evolveu/objects/src/components. Name the files 'account..'.
- write your tests, and Problem Solving Code (PSC) in the correct order

At this point you should have a well tested "Account" class. You will have created an instance of the class to test. That instance of the class is also called an object.

Do a code review with your group. Consider all the different approaches and as a group select the best approach. Consider:

- simplest design
- readability
- function
- speed

Make any changes to your code; you like to have the best code you can write. As Zac would say "make sure you love your code".

Exercise - Create the Account user interface

Now let's start considering React's version of state.

Let's start:

- create a component for your user interface to use the Account class in evolveu/objects/src/components. Make sure to do small steps at a time
- update your app to make one of the icons use your new Account component
- create a simple wireframe to interact with your account that will allow you to execute all your methods
- share with your team; do code reviews and be happy

Exercise - Create the Account Controller

Having an Account class that we can deposit and withdraw from is great, it's just great. But we have a little problem! Most people have multiple accounts.

Let's talk about what type of functionality we need:

- We need the ability to have multiple accounts
- We need the ability to add and remove accounts
- We need the ability to name our accounts
- We need the ability to get the total of our accounts
- We need the ability to know our highest value account (well, I want that so you can practise with logic)
- We need the ability to know our lowest value account (same as above)

Let's make a few enhancements to allow that functionality. Do not change any of the user interface or React code until we have the Plain Old JavaScript Objects (POJOs) working:

- update the Account class to also have a name attribute remembering to test first
- create a new class called Accounts that will be our controller and keep track of all of our accounts
- create functions (test first) that will allow all the functions we have talked about

Once the POJO are working let's get the user interface working:

- create a wireframe to allow users to interact with the system. Maybe each account should be a card UI? Be creative and funky. Brainstorm ideas with your group.
- build your creation, have fun with it
- once you are complete, find Larry and show him. If you can't find him, send him an email and tell him you have the account controller done and in the assignment it told you to send him an email to come and have a look. Great progress.

Look up the term "Module View Controller". Creating tested code that is well architected builds systems that are maintainable and easy to use. Congratulations! You are on a great path.

Exercise - Community and City

We will be creating two classes City and Community along with the appropriate test classes. Continue to create them in evolveu/objects/src/components. City is considered the worker class and Community is the controller class. Build this using TDD.

- create a City class with four attributes on the constructor: Name, Latitude, Longitude, Population
- create a method "show" which creates a single string with the 4 attributes
- create a method "movedIn" which receives a number that will be added to the city's population
- create a method "movedOut" which receives a number that will be subtracted from the city's population
- create a method "howBig" that will return one of the following:
    a. City – a population > 100,000
    b. Large town – a large town has a population of 20,000 to 100,000
    c. Town – a town has a population of 1,000 to 20,000

    d. Village – larger than a hamlet but smaller than a town
    e. Hamlet – population 1 - 100
- in the controller class create methods:
    a. "whichSphere" to return either the Northern Hemisphere or Southern Hemisphere
    b. getMostNorthern
    c. getMostSouthern
    d. getPopulation ⇒ total for all the cities

Once the Classes have been tested you know the schtick…

- create a wireframe to use the functionality. I really like those cards. Use that or something even better
- create the component to implement the wireframe
- use one of the icons in the app to show this component
- show Larry when you're done
- more important, celebrate your success

Exercise - Object Reference

Let's look at object references. Using the last exercise add a test to investigate this functionality:

- write a test that creates a city "myCity"
- make myFav = myCity
- check the population of both references
- add some population to one of the references
- check the population of both references
- what happened and why
- create a test to make sure it keeps working