Instructions:

The code can be run just like the brief's outline:

```
example = new UnidaysDiscountChallenge(pricingRules);

example.AddToBasket(itemOne);
example.AddToBasket(itemTwo);

result = example.CalculateTotalPrice();

totalPrice = result.Total;
deliveryCharge = result.DeliveryCharge;
overallTotal = totalPrice + deliveryCharge;
```

Where itemOne and itemTwo must be strings of the appropriate items (with any number of characters within the string, but only one string).

Problem Approach:

The problem was tackled by:

- Listing the necessary classes and their properties and methods:
    - UnidaysDiscountChallenge:
        - .AddToBasket()
        - ._ItemList, to store all values added by .AddToBasket()
        - ._pricingRules, to store the pricingRules function input as an argument to the UnidaysDiscountChallenge object, as per the example.
        - ._AcceptedItems, a list storing all the accepted item categories (letters a-e) to check items user adds to basket.
        - .CalculateTotalPrice()
            - Returns TotalPrice object containing Total and DeliveryCharge properties.
    - TotalPrice:
        - .Total, would use the pricingRules function as an argument to calculate the total and attach it to the Total property.
        - .DeliveryCharge, would add 7 to the price, or 0 if .Total is greater than or equal to 50.
- Applied the pricingRules through a function, pricingRules(), which takes a list (in this case the UnidaysDiscountChallenge._ItemList property) as an argument and returns the total (in this case it would be used to return TotalPrice.Total, which would then be used to determine TotalPrice.DeliveryCharge).
- Produced a skeleton code of the above to outline the code's structure, including what variables would be needed and what parameters the function and methods would need.
- Pseudocode was developed for each class and function (including methods).
- Found that the TotalPrice class would require 2 arguments: the pricingRules function and the UndiaysDiscountChallenge._ItemList property, so that it could be passed onto the pricingRules function as an argument within the object. This could be avoided and simplified by removing the TotalPrice object and instead use the pseudocode developed for TotalPrice in UnidaysDiscountChallenge.CalculateTotalPrice() directly, which can then access the ._ItemList property from within the UnidaysDiscountChallenge object itself, and only take the pricingRules function as an argument, all while reducing the number of structures necessary in the program.
- UnidaysDiscountChallenge.CalculateTotalPrice() was changed to take in pricingRules as an argument and return a class-less object with Total and DeliveryCharge properties.
- Added error codes and fault-tolerance:
    - Program could take a string of values, rather than just one at a time.
    - Program could take uppercase and lowercase values, and process both of these.
    - Error codes in case of calling .CalculateTotalPrice() when the ._ItemList array is empty, or if the .Total seems.
    - Error code in case the pricingRules() function fails and ends up returning 0 or a negative number.
    - Error code in case a non-string value is entered for .AddToBasket()

- Lastly, the code was tested with all the example tests run in a loop through the program, as well as personal tests adding non-acceptable items (like “F”, “G”, “H”), which were rejected appropriately, and all results were acceptable.

URL and Code Access:
Access and run the code in my repls: https://repl.it/@ehsc1997/UnidaysChallenge-1
or use the attached UnidaysChallenge.js file to run on a JavaScript console, IDE, or website.