Collections Lab

In our development of the backend process, we have come across the requirement that a user be able to add, remove and change products to their shopping carts. Let's write that ability now. And let's do it using test-driven development.

Add to the cart

1. In a test project, add or open a class called "CartTests". In it create a test for a new AddItem() method. This public method should receive two parameters; a Product (Not a ProductId) and an integer -- the quantity of that item being added to the cart.
2. In that test, use Cart.AddItem(Product, Quantity) to add some items to the cart and check the subtotal. Assert that the manually-calculated subtotal is equal to the subtotal returned by cart.Subtotal().
3. Build your solution. Did it compile? _____ It shouldn't have because you haven't written your AddItem method yet.
4. Create an AddItem() method stub in the Cart class. Have it throw a new NotImplementedException.
5. Run and test. Your solution should build but your unit test should fail. There's your red light.
6. Now make the unit test turn green. Replace your NotImplementedException with some working code that actually adds OrderDetails to your cart object.
7. Re-run your unit test. If the light turns green, you can go on.
8. Did you make it a naïve implementation? If so, go back in and refactor it.

Clear the cart

9. Create a new test "CanClearCart()" It should make sure you're able to, well, clear the cart. It should call a method on a Cart object called Clear().
10. Write a method stub in Cart.
11. Run your test to watch the light turn red.
12. Now make your test turn green using a naïve implementation.

Remove a line from the cart

13. Ask your partner what we should name a method that tests to see if we can remove a line item from the cart.
14. Go write that test method!
15. Write a method stub in Cart to remove a line item. This method should receive a ProductId (not a product). It should find the first line in the cart for that productId and remove it from the collection of OrderDetails. Hint: here's a concise way to remove:

```
OrderDetails.Remove(OrderDetails.First(od => od.ProductId == ProductId));
```

16. Write and re-test until that light and all the other lights turn green.
17. Refactor!

Update a quantity in the cart

18. One last requirement is missing from the Cart class; updating a quantity in the cart.
19. Write a test to see if we can update a quantity of an existing item in the cart.
20. You know the drill by now ... Write the method stub so the solution will build. Then write code until the light turns green. Then refactor. Hint: use a foreach to loop through the

OrderDetails in the cart. When you find the line whose ProductId matches the one passed in to this method, change it's quantity.

Bonus!! You and your partner think about these situations:

- What if the product doesn't exist in this cart? What should happen?
- What if the quantity is goofy? Like negative numbers or ridiculously large numbers?
- What if we set the quantity to zero? Should we just remove the whole line?

If you have time, write tests for these situations and handle them in the Cart class. Make the tests turn green.