Steps:

1. Discuss with the client about high level requirements, just like bulleted points.

* Add one or more items to the cart
* Remove an item from the cart
* Add more than one units of same item to the cart
* View total number of items in the cart
* View listing of all items in the cart with their price. Get order total (i.e. price \* quantity)

1. Class elicitation: Here we identify some of the basic entities\classes which participate in the business.

Ex:

ShoppingCart

Item

-Book

-Calendar

-Pen

-TShirt

Here Item can be converted into an enum, like

Item (Book, Calendar, Pen, TShirt)

We can associate quantity with and type of item with an entity LineItem

So we identified the basic entities needed. Let’s attach some properties to them.

ShoppingCart – It’s just a composer of other objects

Item – Id, Name, Price, ItemType(Book, Calendar, Pen, TShirt)

LineItem – ItemId, Quantity

Step 2:

Create these skeleton classes first. Don’t add any properties or methods yet.

Step 3:

Start creating unit tests. First let’s start with ShoppingCart class. We try to add an item to the cart. Then we will assert the no. of items added to the cart.

Test1: should\_add\_an\_item\_to\_the\_cart()

[Fact]

public void should\_add\_an\_item\_to\_the\_cart()

{

//arrange

ShoppingCart cart = new ShoppingCart();

//act

cart.addItem(new Item("item-123", "Effective Java", ItemType.Book, 4000));

int totalCount = cart.GetTotalNumberOfItems();

//assert

Assert.Equal(1, totalCount);

}

Now let’s add the enum ItemType

Then add Item class properties. You can define the private parameterized constructor with these parameters to it. We will keep this private, because we introduce a builder for Item class construction. (Fluent API).

The Item class must have all the properties defined, with a public default constructor (builder will use this default constructor to create an empty Item object initially).

public class Item

{

public string Id { get; set; }

public string Name { get; set; }

public ItemType ItemType { get; set; }

public decimal Price { get; set; }

public Item() { }

}

Now define a class ItemBuilder. This will have a private instance of Item and “build()” method which returns the this Item instance (at the end of object construction). Now define multiple fluent api methods to set the properties of Item like Id, Name, Price etc…

public class ItemBuilder

{

private Item item = new Item();

public ItemBuilder() { }

public Item build() => item;

public ItemBuilder withId(string id)

{

item.Id = id;

return this;

}

public ItemBuilder withName(string name)

{

item.Name = name;

return this;

}

public ItemBuilder withItemType(ItemType type)

{

item.ItemType = type;

return this;

}

public ItemBuilder withPrice(decimal price)

{

item.Price = price;

return this;

}

}

Now we define void method AddItem() and int method GetTotalNumberOfItems() on ShoppingCart class and run the tests.

This test will fail because it expects 1 but returns 0;

Let’s fix it by adding a PRIVATE collection of Item shopping cart class.

private IList<Item> ItemList = new List<Item>();

Fix AddItem() method to add item to the list and GetTotalNumberOfItems() method to return list count, so that test can pass.

Test 2: should\_add\_multiple\_items\_to\_the\_cart(), make sure it just passes.

Test 3: should\_add\_multiple\_quantity\_of\_same\_item\_to\_the\_cart()

We write this test where we add multiple items of same type to the cart and check its behaviour.

Ex: 2 Books, 3 Pens etc…

For this we will introduce a new class, which represent a particular item with its quantity.

public class LineItem

{

public Item Item { get; private set; }

public int Quantity { get; private set; }

public LineItem(Item item, int quantity = 1)

{

Item = item;

Quantity = quantity;

}

}

Note that by default the Quantity will be set to 1.

Now we will make our ShoppingCart to have the collection of LineItem instead of Item. This will force us to modify the previous tests also.

Let’s modify all the tests along with ShoppingCart and then rerun them. First 2 tests will pass, but this test will fail, this is because we need to get the quantity of each line item and add. So let’s modify the ShoppingCart’s TotalNumberOfItems() method for this.

public int GetTotalNumberOfItems()

{

int totalCount = 0;

foreach(var lItem in ItemList)

{

totalCount += lItem.Quantity;

}

return totalCount;

}

Now, run all the tests and make sure all runs successfully.

Test 4: Remove item from the cart

Test 5: Display order\cart total