HOTFIX- BALANCEO promociones directas + promociones de origis + bonificaciones con monederos de empleado

Autor: Huilver Nolasco Aguilar

Fecha:Octubre del 2024

Contenido

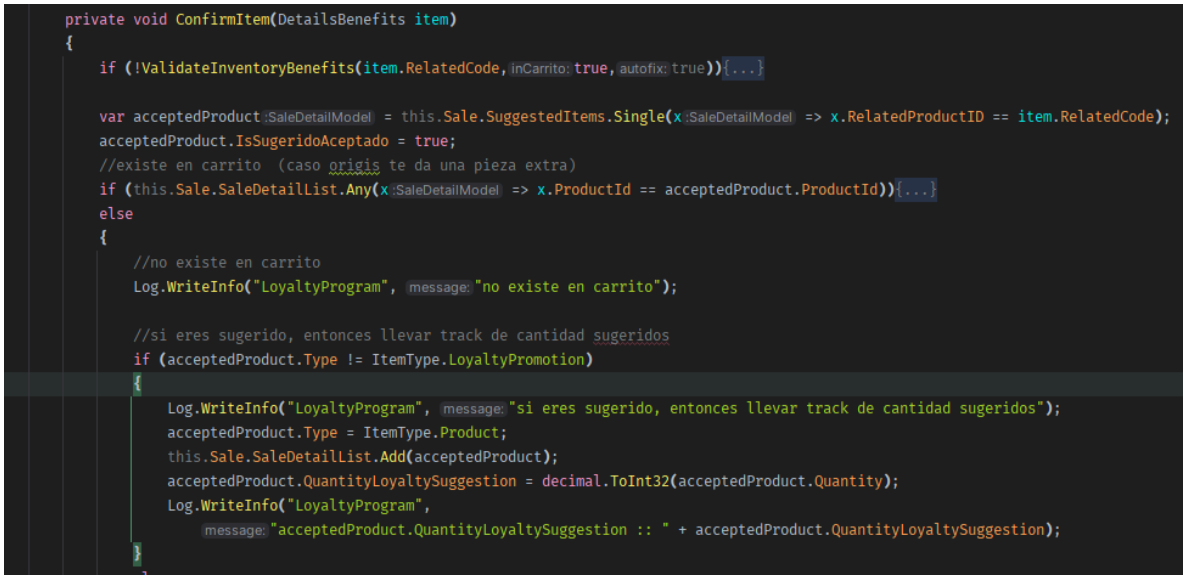
Descripción general

Actualmente existe un incidente con el cálculo de la venta, respecto a las promociones, que se produce cuando se tienen promociones directas con de productos y estos tienen promociones de origis, esto generalmente en monederos para empleados de farmacias.

Causa

En balanceo existen varios caminos dependiendo de lo que origis y el motor de promociones dicta, uno de ellos deriva en que cuando se se llega a la confirmación del producto sugerido que no forma parte de la venta, si no que viene como regalo pendiente asociado al monedero. Bajo estas condiciones entra un lugar del código en donde el sugerido pasa a formar parte de la lista de productos del carro de compras, pero al hacerlo no lleva un dato, el cual es un detonante para el recálculo de los precios del producto y en general para la venta completa.

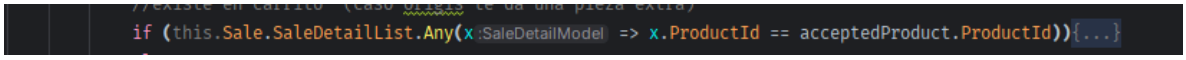
Para mostrar esto se muestra la siguiente imagen de código que actualmente está operando esta lógica



```
private void ConfirmItem(DetailsBenefits item)
{
    if (!ValidateInventoryBenefits(item.RelatedCode, inCarrito: true, autofix: true)){...}

    var acceptedProduct :SaleDetailModel = this.Sale.SuggestedItems.Single(x :SaleDetailModel => x.RelatedProductID == item.RelatedCode);
    acceptedProduct.IsSugeridoAceptado = true;
    //existe en carrito  (caso origis te da una pieza extra)
    if (this.Sale.SaleDetailList.Any(x :SaleDetailModel => x.ProductId == acceptedProduct.ProductId)){...}
    else
    {
        //no existe en carrito
        Log.WriteInfo("LoyaltyProgram", message: "no existe en carrito");

        //si eres sugerido, entonces llevar track de cantidad sugeridos
        if (acceptedProduct.Type != ItemType.LoyaltyPromotion)
        {
            Log.WriteInfo("LoyaltyProgram", message: "si eres sugerido, entonces llevar track de cantidad sugeridos");
            acceptedProduct.Type = ItemType.Product;
            this.Sale.SaleDetailList.Add(acceptedProduct);
            acceptedProduct.QuantityLoyaltySuggestion = decimal.ToInt32(acceptedProduct.Quantity);
            Log.WriteInfo("LoyaltyProgram",
                message: "acceptedProduct.QuantityLoyaltySuggestion :: " + acceptedProduct.QuantityLoyaltySuggestion);
        }
```

A continuación la explicación de código:

Se pregunta si el producto sugerido ya viene en el carrito de compras

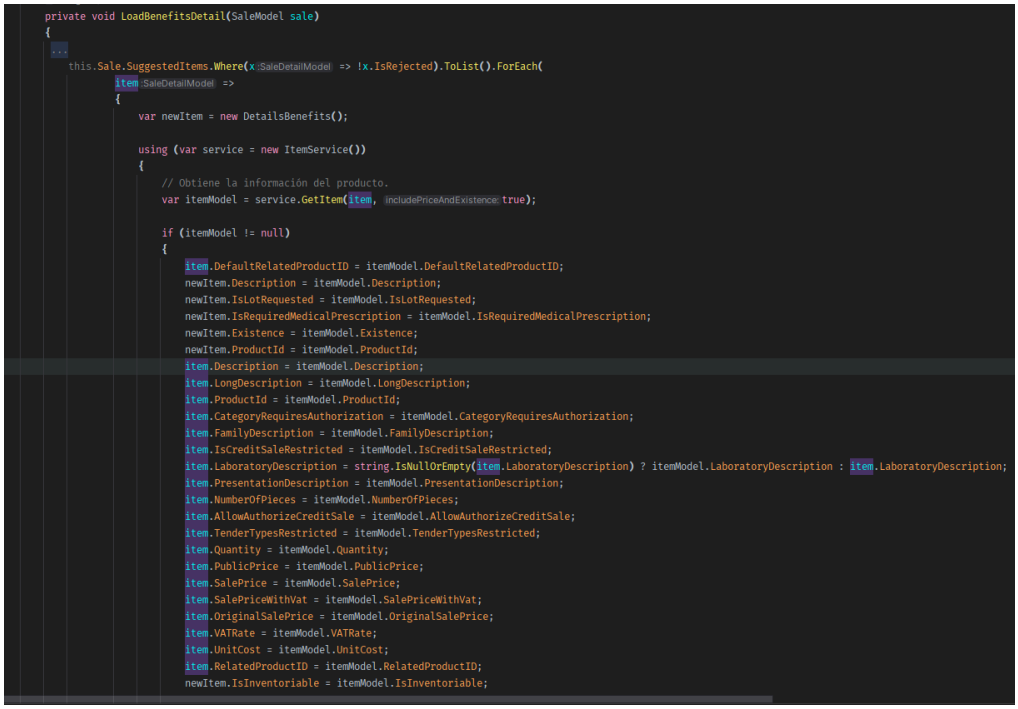


Si no existe el producto sugerido aceptado en el carro de compras, se agrega al carro de compras:

```
else
{
    //no existe en carrito
    Log.WriteInfo("LoyaltyProgram", message: "no existe en carrito");

    //si eres sugerido, entonces llevar track de cantidad sugeridos
    if (acceptedProduct.Type != ItemType.LoyaltyPromotion)
    {
        Log.WriteInfo("LoyaltyProgram", message: "si eres sugerido, entonces llevar track de cantidad sugeridos");
        acceptedProduct.Type = ItemType.Product;
        this.Sale.SaleDetailList.Add(acceptedProduct);
        acceptedProduct.QuantityLoyaltySuggestion = decimal.ToInt32(acceptedProduct.Quantity);
        Log.WriteInfo("LoyaltyProgram",
            message: "acceptedProduct.QuantityLoyaltySuggestion :: " + acceptedProduct.QuantityLoyaltySuggestion);
    }
```

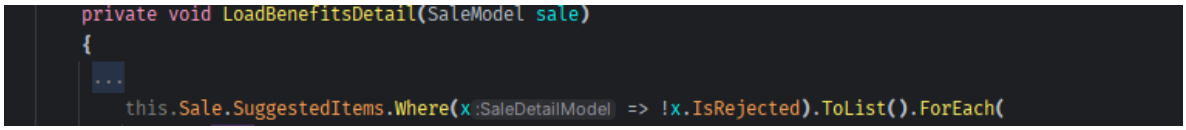
Sin embargo este producto aceptado no trae una propiedad que se usa en una condición posterior, la propiedad es la de `CodeDepartment`

y esto se puede observar en el siguiente código, que se dispara al mostrar el producto sugerido:



```
private void LoadBenefitsDetail(SaleModel sale)
{
...
    this.Sale.SuggestedItems.Where(x :SaleDetailModel => !x.IsRejected).ToList().ForEach(
            item :SaleDetailModel =>
            {
                var newItem = new DetailsBenefits();

                using (var service = new ItemService())
                {
                    // Obtiene la información del producto.
                    var itemModel = service.GetItem(item, includePriceAndExistence: true);

                    if (itemModel != null)
                    {
                        item.DefaultRelatedProductID = itemModel.DefaultRelatedProductID;
                        newItem.Description = itemModel.Description;
                        newItem.IsLotRequested = itemModel.IsLotRequested;
                        newItem.IsRequiredMedicalPrescription = itemModel.IsRequiredMedicalPrescription;
                        newItem.Existence = itemModel.Existence;
                        newItem.ProductId = itemModel.ProductId;
                        item.Description = itemModel.Description;
                        item.LongDescription = itemModel.LongDescription;
                        item.ProductId = itemModel.ProductId;
                        item.CategoryRequiresAuthorization = itemModel.CategoryRequiresAuthorization;
                        item.FamilyDescription = itemModel.FamilyDescription;
                        item.IsCreditSaleRestricted = itemModel.IsCreditSaleRestricted;
                        item.LaboratoryDescription = string.IsNullOrEmpty(item.LaboratoryDescription) ? itemModel.LaboratoryDescription : item.LaboratoryDescription;
                        item.PresentationDescription = itemModel.PresentationDescription;
                        item.NumberOfPieces = itemModel.NumberOfPieces;
                        item.AllowAuthorizeCreditSale = itemModel.AllowAuthorizeCreditSale;
                        item.TenderTypesRestricted = itemModel.TenderTypesRestricted;
                        item.Quantity = itemModel.Quantity;
                        item.PublicPrice = itemModel.PublicPrice;
                        item.SalePrice = itemModel.SalePrice;
                        item.SalePriceWithVat = itemModel.SalePriceWithVat;
                        item.OriginalSalePrice = itemModel.OriginalSalePrice;
                        item.VATRate = itemModel.VATRate;
                        item.UnitCost = itemModel.UnitCost;
                        item.RelatedProductID = itemModel.RelatedProductID;
                        newItem.IsInventoriable = itemModel.IsInventoriable;
```

Para todos los productos sugeridos que no hayan sido aceptados:



```
private void LoadBenefitsDetail(SaleModel sale)
{
...
    this.Sale.SuggestedItems.Where(x :SaleDetailModel => !x.IsRejected).ToList().ForEach(
```

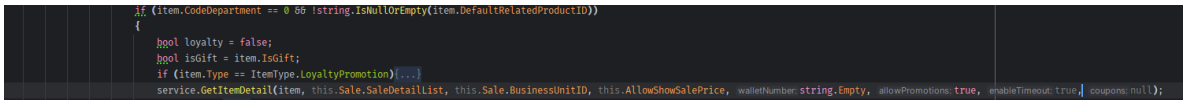
Completa la información del producto sugerido no aceptado:

```
var itemModel = service.GetItem(item, includePriceAndExistence: true);

if (itemModel != null)
{
    item.DefaultRelatedProductID = itemModel.DefaultRelatedProductID;
    newItem.Description = itemModel.Description;
    newItem.IsLotRequested = itemModel.IsLotRequested;
    newItem.IsRequiredMedicalPrescription = itemModel.IsRequiredMedicalPrescription;
    newItem.Existence = itemModel.Existence;
    newItem.ProductId = itemModel.ProductId;
    item.Description = itemModel.Description;
    item.LongDescription = itemModel.LongDescription;
    item.ProductId = itemModel.ProductId;
    item.CategoryRequiresAuthorization = itemModel.CategoryRequiresAuthorization;
    item.FamilyDescription = itemModel.FamilyDescription;
    item.IsCreditSaleRestricted = itemModel.IsCreditSaleRestricted;
    item.LaboratoryDescription = string.IsNullOrEmpty(item.LaboratoryDescription) ? itemModel.LaboratoryDescription : item.LaboratoryDescription;
    item.PresentationDescription = itemModel.PresentationDescription;
    item.NumberOfPieces = itemModel.NumberOfPieces;
    item.AllowAuthorizeCreditSale = itemModel.AllowAuthorizeCreditSale;
    item.TenderTypesRestricted = itemModel.TenderTypesRestricted;
    item.Quantity = itemModel.Quantity;
    item.PublicPrice = itemModel.PublicPrice;
    item.SalePrice = itemModel.SalePrice;
    item.SalePriceWithVat = itemModel.SalePriceWithVat;
    item.OriginalSalePrice = itemModel.OriginalSalePrice;
    item.VATRate = itemModel.VATRate;
    item.UnitCost = itemModel.UnitCost;
    item.RelatedProductID = itemModel.RelatedProductID;
    newItem.IsInventoriable = itemModel.IsInventoriable;
```

Sin embargo no está considerado `CodeDepartment`, que hace que se dispare un recálculo de precios del producto sugerido y por consecuencia un recálculo de la venta, debido a que un cambio en el carro de compras siempre implica un cálculo de totales

```
private ResultStruct<bool> UpSertSale(SaleModel sale, string status = SaleStatus.New)
{
    Log.Write("Ejecutando UpSertSale - estatus: " + status);

    List<LotModel> newLots = sale.LotSaleDetail.Lots.Where(x:LotModel => !x.IsPersisted).ToList();
    if (newLots.Any()){...}

    if (sale.Status != SaleStatus.Pending){...}

    // Permite recordar al vendedor que realizo una preventa con anticipo
...

    var promotionIds:List<SaleDetailModel> = sale.SaleDetailList.Where(f:SaleDetailModel => (sale.CouponInformation != null && sale.CouponInformation.Any(x:CouponInformationModel => x.PromotionId != f.PromotionId) &&

    if (sale.SaleDetailList.Any(x:SaleDetailModel => x.CodeDepartment == 0))
    {
        if (sale.SaleType == EnumSaleType.Credit && sale.SaleDetailList.All(x:SaleDetailModel => x.CodeDepartment == 0)){...}

        using (var service = new SaleService())
        {
            foreach (SaleDetailModel item in sale.SaleDetailList)
            {
                if (item.CodeDepartment == 0 && !string.IsNullOrEmpty(item.DefaultRelatedProductID))
                {
                    bool loyalty = false;
                    bool isGift = item.IsGift;
                    if (item.Type == ItemType.LoyaltyPromotion){...}
                    service.GetItemDetail(item, this.Sale.SaleDetailList, this.Sale.BusinessUnitID, this.AllowShowSalePrice, walletNumber:string.Empty, allowPromotions:true, enableTimeout:true, coupons:null);
                    if (loyalty){...}
                }
            }
        }
```

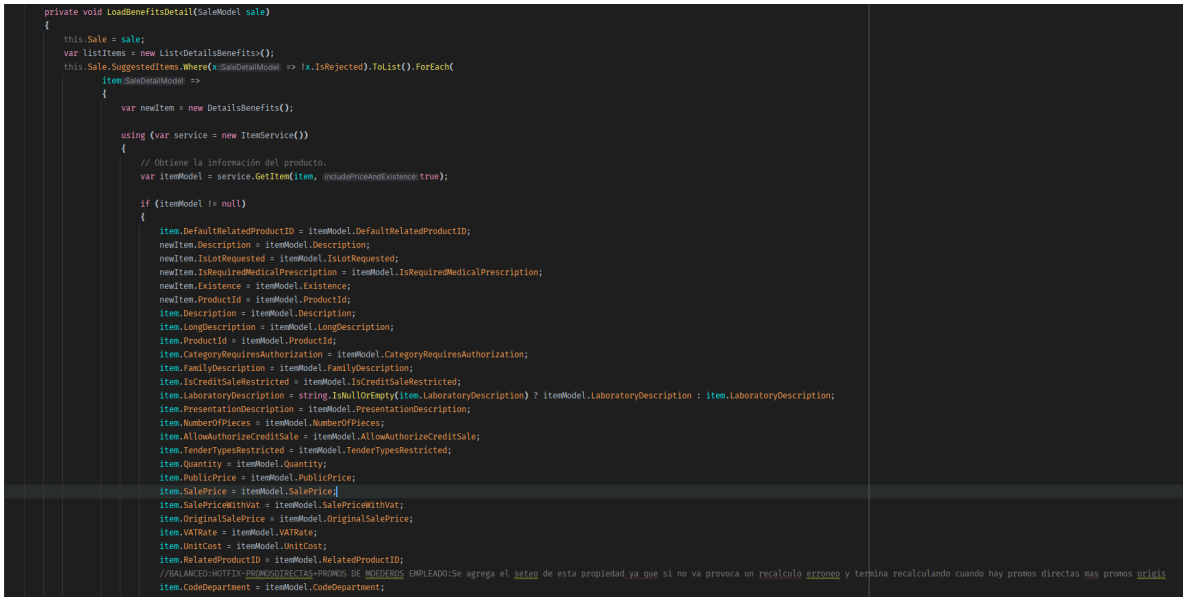
Aquí es donde se realiza dicho recálculo de precios del producto aceptado, sin embargo para este punto el cálculo de las promociones directas + origis + sugeridos aceptados ya viene calculada



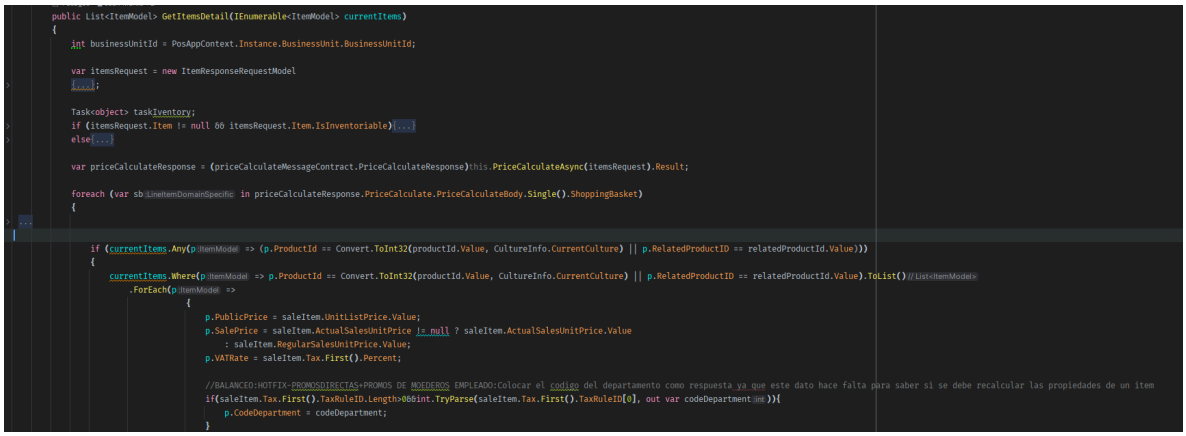
Hacerlo en este punto con el estado actual de la venta provoca un cálculo incorrecto.

Solución

La solución consiste en colocar el dato de `CodeDepartment` al cargar la pantalla de aceptación o rechazo del producto sugerido, de tal manera que en los flujos subsecuentes no se haga un cálculo incorrecto de la venta:

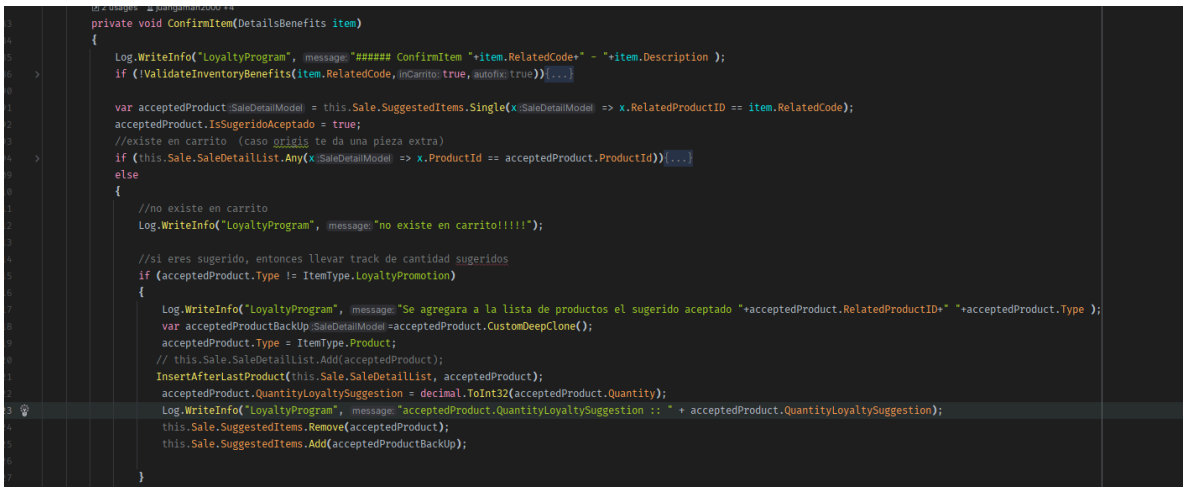


Sin embargo el método actual de obtención de atributos del producto, no considera el dato del CodeDepartment, por lo que es necesario considerarlo:



```
public List<ItemModel> GetItemsDetail(IEnumerable<ItemModel> currentItems)
{
    int businessUnitId = PosAppContext.Instance.BusinessUnit.BusinessUnitId;

    var itemsRequest = new ItemResponseRequestModel
    {...};

    Task<object> taskIventory;
    if (itemsRequest.Item != null && itemsRequest.Item.IsInventoriable){...}
    else{...}

    var priceCalculateResponse = (priceCalculateMessageContract.PriceCalculateResponse)this.PriceCalculateAsync(itemsRequest).Result;

    foreach (var sb:LineItemDomainSpecific in priceCalculateResponse.PriceCalculate.PriceCalculateBody.Single().ShoppingBasket)
    {
...

        if (currentItems.Any(p:ItemModel => (p.ProductId == Convert.ToInt32(productId.Value, CultureInfo.CurrentCulture) || p.RelatedProductID == relatedProductId.Value)))
        {
            currentItems.Where(p:ItemModel => p.ProductId == Convert.ToInt32(productId.Value, CultureInfo.CurrentCulture) || p.RelatedProductID == relatedProductId.Value).ToList()//List<ItemModel>
                .ForEach(p:ItemModel =>
                            {
                                p.PublicPrice = saleItem.UnitListPrice.Value;
                                p.SalePrice = saleItem.ActualSalesUnitPrice != null ? saleItem.ActualSalesUnitPrice.Value
                                    : saleItem.RegularSalesUnitPrice.Value;
                                p.VATRate = saleItem.Tax.First().Percent;

                                //BALANCED:HOTFIX-PROMOSDIRECTAS+PROMOS DE MOEDEROS EMPLEADO:Colocar el codigo del departamento como respuesta ya que este dato hace falta para saber si se debe recalcular las propiedades de un item
                                if(saleItem.Tax.First().TaxRuleID.Length>0&&int.TryParse(saleItem.Tax.First().TaxRuleID[0], out var codeDepartment:int)){
                                    p.CodeDepartment = codeDepartment;
                                }
```

Además al agregar el producto sugerido, se coloca en el carro de compras justo despues del ultimo producto y se deja intacto el producto que viene en la lista de sugeridos para evitar cualquier tipo recálculo posterior asociado a un cambio realizado:



```
private void ConfirmItem(DetailsBenefits item)
{
    Log.WriteInfo("LoyaltyProgram", message:"###### ConfirmItem "+item.RelatedCode+" - "+item.Description );
    if (!ValidateInventoryBenefits(item.RelatedCode,inCarrito:true,autofix:true)){...}

    var acceptedProduct:SaleDetailModel = this.Sale.SuggestedItems.Single(x:SaleDetailModel => x.RelatedProductID == item.RelatedCode);
    acceptedProduct.IsSugeridoAceptado = true;
    //existe en carrito  (caso origis te da una pieza extra)
    if (this.Sale.SaleDetailList.Any(x:SaleDetailModel => x.ProductId == acceptedProduct.ProductId)){...}
    else
    {
        //no existe en carrito
        Log.WriteInfo("LoyaltyProgram", message:"no existe en carrito!!!!!");

        //si eres sugerido, entonces llevar track de cantidad sugeridos
        if (acceptedProduct.Type != ItemType.LoyaltyPromotion)
        {
            Log.WriteInfo("LoyaltyProgram", message:"Se agregara a la lista de productos el sugerido aceptado "+acceptedProduct.RelatedProductID+" "+acceptedProduct.Type );
            var acceptedProductBackUp:SaleDetailModel =acceptedProduct.CustomDeepClone();
            acceptedProduct.Type = ItemType.Product;
           // this.Sale.SaleDetailList.Add(acceptedProduct);
           InsertAfterLastProduct(this.Sale.SaleDetailList, acceptedProduct);
            acceptedProduct.QuantityLoyaltySuggestion = decimal.ToInt32(acceptedProduct.Quantity);
            Log.WriteInfo("LoyaltyProgram", message:"acceptedProduct.QuantityLoyaltySuggestion :: " + acceptedProduct.QuantityLoyaltySuggestion);
            this.Sale.SuggestedItems.Remove(acceptedProduct);
            this.Sale.SuggestedItems.Add(acceptedProductBackUp);

        }
```