Motivation:

Break method is like an extract method that can decompose one large method to several smaller methods. In this example shows a large method which is AcceptPayment method. There are lots of supporting code working in this method. So, I will use the break method to decompose the AcceptPayment method to smaller method.

**Before:**

public class CashRegister

{

public CashRegister()

{

Tax = 0.06m;

}

private decimal Tax { get; set; }

/// <summary>

/// This Method doing lots of logical calculations

/// </summary>

/// <param name="customer"></param>

/// <param name="products"></param>

/// <param name="payment"></param>

public void AcceptPayment(Customer customer, IEnumerable<Product> products, decimal payment)

{

decimal subTotal = 0m;

foreach (Product product in products)

{

subTotal += product.Price;

}

foreach (Product product in products)

{

subTotal -= product.AvailableDiscounts;

}

decimal grandTotal = subTotal \* Tax;

customer.DeductFromAccountBalance(grandTotal);

}

}

public class Customer

{

public void DeductFromAccountBalance(decimal amount)

{

// deduct from balance

}

}

public class Product

{

public decimal Price { get; set; }

public decimal AvailableDiscounts { get; set; }

}

**Mechanics:**

Here you can see the CashRegister class has a method named AcceptPayment. There are lots of supporting code in this method. These codes are working for this methods functionality. Using break method refactoring technique I am going to extract method based on different calculations. In this method there are calculation of subtotal, discount, TAX, and balanceDeduction calculation. These calculations can be move to four method. So, extract these four methods from the AcceptPayment method.

**After:**

public class CashRegister

{

public CashRegister()

{

Tax = 0.06m;

}

private decimal Tax { get; set; }

private IEnumerable<Product> Products { get; set; }

/// <summary>

/// This method using for accepting payment

/// </summary>

/// <param name="customer"></param>

/// <param name="products"></param>

/// <param name="payment"></param>

public void AcceptPayment(Customer customer, IEnumerable<Product> products, decimal payment)

{

decimal subTotal = CalculateSubtotal();

subTotal = SubtractDiscounts(subTotal);

decimal grandTotal = AddTax(subTotal);

SubtractFromCustomerBalance(customer, grandTotal);

}

/// <summary>

/// This method calculates the deducted balance

/// </summary>

/// <param name="customer"></param>

/// <param name="grandTotal"></param>

private void SubtractFromCustomerBalance(Customer customer, decimal grandTotal)

{

customer.DeductFromAccountBalance(grandTotal);

}

/// <summary>

/// This method calculates the TAX based on subtotal

/// </summary>

/// <param name="subTotal"></param>

/// <returns></returns>

private decimal AddTax(decimal subTotal)

{

return subTotal \* Tax;

}

/// <summary>

/// This method calculate discount

/// </summary>

/// <param name="subTotal"></param>

/// <returns></returns>

private decimal SubtractDiscounts(decimal subTotal)

{

foreach (Product product in Products)

{

subTotal -= product.AvailableDiscounts;

}

return subTotal;

}

/// <summary>

/// This method calculates the subtotal of the product price.

/// </summary>

/// <param name="customer"></param>

/// <param name="grandTotal"></param>

private decimal CalculateSubtotal()

{

decimal subTotal = 0m;

foreach (Product product in Products)

{

subTotal += product.Price;

}

return subTotal;

}

}

public class Customer

{

public void DeductFromAccountBalance(decimal amount)

{

// deduct from balance

}

}

public class Product

{

public decimal Price {get; set;}

public decimal AvailableDiscounts {get; set;}

}

Here AcceptPayment method divided into four methods called **CalculateSubtotal(), CalculateSubtotal(),SubtractDiscounts(),SubtractFromCustomerBalance()**. This is how I refactored the code using break method.