# Inner Classes for Component-Oriented Programming: an Example (3)

```
import java.util.*;

// Test code:
// ----------
// ----------

 class Client {

   static public void main(String[] args) {

      Tax t1 = new Tax((float)0.05);
      Tax t2 = new Tax((float) 0.07);
      Tax[] t = {t1, t2};

      HWProduct product =
          new HWProduct("Schreibmaschine", (float) 250, (float) 200, t);
      Customer cust = new Customer("Schmidt", (float) 0.5, (float) 0.02);

      Quote q = new Quote(product, cust, 10);
      RegScheme regSchemeAdapapter = RegScheme.singleInstance();
      System.out.println("Reg. Price = " + regSchemeAdapapter.setHost(q).price());
      System.out.println(q.test);

//    ...

// Later after I have added negotiated pricing
       System.out.println("Changing the scheme ...");
```

```
      NegScheme negSchemeAdapapter = NegScheme.singleInstance();
      System.out.println("Neg. Price = " + negSchemeAdapapter.setHost(q).price());
      System.out.println(q.test);

   }
 }

// Result:
// -------
// Reg. Price = 224.0
// Hello
// Changing the scheme ...
// Neg. Price = 112.0
// Hello


// -----------------------------------------------------------------------------
// -----------------------------------------------------------------------------

// APPLICATION CODE
// ----------------

 class HWProduct {

    protected String name;
    protected float price, salePrice;
    protected Tax[] taxes;
    // protected HashTable discount;
    // protected Producer producer;
    // protected Date entered;

    Tax[] taxes() {return taxes;}
    float salePrice() {return salePrice;}
    float saleDiscount() {return 0;}
    float regPrice() {return price;}
    // float regDiscount(int qty) {return discount.lookup(qty);}
    float regDiscount(int qty) {return qty * (float) 0.02;}

    // Producer producer() {retrun producer;}
    public HWProduct() {}
    public HWProduct (String prodName, float pr, float salePr, Tax[] tax) {
       name = prodName;
       price = pr;
       salePrice = salePr;
```

```
      taxes = tax;
    }

}


class Tax {
   protected float salePercentage;
   // protected HashTable productCategories;

   float taxCharge(HWProduct prod, float unitPr, int qty) {
      // HashKey key = new HashKey(prod, qty);
      // float categPercentage = productCategories.lookup(key);
      // return unitPr * categPercentage * salePercentage;
         return unitPr * salePercentage;
    }

    public Tax(float perc) { salePercentage = perc; }
}

 class Quote {

   protected HWProduct prod;
   protected int qty;
   protected Customer cust;
   // protected Employee enteredBy;
   public String test = "Hello";
   // protected Date deliveryTime;
   // protected Date orderEntryTime;

   int quantity() {return qty;}
   HWProduct prod() {return prod;}
   Customer customer() {return cust;}

public Quote() {};


   public Quote(HWProduct product, Customer customer, int quantity) {
     cust = customer;
     prod = product;
     qty = quantity;
    }
 }
```

```
  class Customer {

    protected String name;
    // protected String address;
    // protected HashTable negotiatedProdPrices;
    protected float negotiatedProdPrices;
    protected float negotiatedDiscountPercentage;

    // float negProdPrice(HWProduct prod) {
    //     negotiatedProdPrices.lookup((Object) prod);
    //  }

    float negProdPrice(HWProduct prod) {
        float pr = prod.regPrice();
        return (pr - pr * negotiatedProdPrices);
     }

    float negProdDiscount(HWProduct prod, int qty) {
       return qty * negotiatedDiscountPercentage;
      }

    String getName() {return name;}

    public Customer(String custName, float negProdPrices, float negDiscPerc) {
      name = custName;
      negotiatedProdPrices = negProdPrices;
      negotiatedDiscountPercentage = negDiscPerc;
     }
   }

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------

// PRICING COMPONENT
// -----------------

 abstract class LineItemParty {
      abstract int quantity();
      abstract ItemParty item();
      abstract PricerParty pricer();
      abstract CustomerParty customer();

      public float price() {
```

```
        float discount;
        float basicPrice, unitPrice;
        PricerParty pricer = pricer();
        CustomerParty customer = customer();
        int qty = quantity();
        ItemParty item = item();
        basicPrice = pricer.basicPrice(item);
        discount = pricer.discount(item, qty, customer);
        unitPrice = basicPrice - (discount * basicPrice);
        return unitPrice + item.addCharges(unitPrice, qty);
    }
  }


 abstract class ItemParty {

    float addCharges(float unitPrice, int qty) {
       float total = 0;
       for (int i = 0; i < charges().length; i++)
            total += charges()[i].cost(qty, unitPrice, this);
       return total;
     }

   abstract ChargerParty[] charges();
 }


 interface PricerParty {
    float basicPrice(ItemParty item);
    float discount(ItemParty item, int qty, CustomerParty cust);
  }

 interface ChargerParty {
    float cost (int qty, float unitPr, ItemParty item);
  }

 interface CustomerParty {}



// -----------------------------------------------------------------------
// -----------------------------------------------------------------------

// Adaptation Code:
```

```
// ----------------
// ----------------

interface AdapterObject {
   Object enclosed();
 }

abstract class AbstractPricingScheme {

   Quote host;
   public AbstractPricingScheme setHost(Quote q) {
         host = q;
         return this;}

   public float price() {return lineItemAdapter().price();}

   public LineItemParty lineItemAdapter() {
     LineItemParty lineItemAdapter = new LineItemParty() {
        protected ItemParty item() {return itemAdapter();}
        protected CustomerParty customer() {return customerAdapter(); }
        protected int quantity() {return host.quantity();}
        public Object enclosed() {return host;}
        protected PricerParty pricer() {return pricerAdapter();}
     };
     return lineItemAdapter;
  }

  abstract public PricerParty pricerAdapter();

  private abstract class MyChargerParty implements ChargerParty, AdapterObject {
    abstract public float cost(int qty, float unitPr, ItemParty item);
    abstract public Object enclosed();
  }

  public ItemParty itemAdapter() {
    class MyItemParty extends ItemParty implements AdapterObject {
       public ChargerParty [] charges() {
         final Tax[] _taxes = host.prod.taxes;
         final int len = _taxes.length;
         ChargerParty [] taxTable = new ChargerParty[len];
         for (int i = 0; i < len; i++) {
            final Tax currTax = _taxes[i];
            ChargerParty chargerAdapter = new MyChargerParty() {
               public float cost(int qty, float unitPr, ItemParty item) {
```

```
                 HWProduct _prod = (HWProduct) ((AdapterObject)item).enclosed();
                 return currTax.taxCharge(_prod, unitPr, qty);
                }
              public Object enclosed() {return currTax;}
             };
           taxTable[i] = chargerAdapter;
         }
        return taxTable;
       }

      public Object enclosed() {return host.prod;}
     }
    ItemParty itemAdapter = new MyItemParty();
    return itemAdapter;
   }

  public CustomerParty customerAdapter() {
    class MyCustomerParty implements CustomerParty, AdapterObject {
           public Object enclosed() {return host.cust;}
    }
    CustomerParty customerAdapter = new MyCustomerParty();
    return customerAdapter;
    }
 }

// ------------------------------------------------------------------------------
// ------------------------------------------------------------------------------

// Regular Scheme
// --------------

class RegScheme extends AbstractPricingScheme {

  static public RegScheme singleInstance = new RegScheme();
  static RegScheme singleInstance() {return singleInstance;}

  public PricerParty pricerAdapter() {
       final HWProduct p = host.prod;
       class MyPricerParty implements PricerParty, AdapterObject {
          public float basicPrice(ItemParty item) {return p.regPrice();}
          public float discount(ItemParty item, int qty, CustomerParty cust) {
                          return p.regDiscount(qty); }
          public Object enclosed() {return p;}
         }
```

```
       PricerParty pricerAdapter = new MyPricerParty();
       return pricerAdapter;
    }

 }

// -------------------------------------------------------------------------------
// -------------------------------------------------------------------------------

// Negotiated Scheme
// -----------------

class NegScheme extends AbstractPricingScheme {

  static public NegScheme singleInstance = new NegScheme();
  static NegScheme singleInstance() {return singleInstance;}


   public PricerParty pricerAdapter() {
    final Customer c = host.cust;
    class MyPricerParty implements PricerParty, AdapterObject {
        public float basicPrice(ItemParty item) {
           return c.negProdPrice((HWProduct) ((AdapterObject) item).enclosed());
        }
        public float discount(ItemParty item, int qty, CustomerParty cust) {
         return c.negProdDiscount((HWProduct) ((AdapterObject)item).enclosed(), qty);
         }
        public Object enclosed() {return c;}
     }
    PricerParty pricerAdapter = new MyPricerParty();
    return pricerAdapter;
  }
}

// ----------------------------------------------------------------------
// ----------------------------------------------------------------------

// Total component
// ---------------

// package Total;
```