Jessica Wei (7952),
Austin Patton (8859),
Jeremy Escamilla (6545),
Mia Keierleber (7531)
CECS 343

Classes & Methods

<u>MAIN CLASS (MENU & PASSWORD INCLUSIVE)</u>

```
class PotatoElectronicsStore
{
     /* Class Description: The main class used to instantiate
     all classes when the user has been authorized for access
     and loop the main menu */
     /*
     Password
          Requests the password from the user until they enter
          the correct password.
     Inputs:
          None
     Outputs:
          None
     Return Value:
          None
     Exceptions:
          InvalidInput - if the entered text does not match the
          stored password./*
     void enterPassword();
     /*
     mainMenu
          Directs the user to another submenu for the
          option/class they chose
     Inputs:
          None
     Outputs:
          None
     Return Value:
          None
     Exceptions:
          InvalidInput - if mainChoice is not one of the listed
          integers on the main menu */
     void mainMenu();
}
```

SALESPERSON

```
class Salesperson
{
     /*Class Description: The salesperson class used to define
     the attributes of a salesperson.*/
     /*
     salespersonConstructor
          Creates the salesperson.
     Inputs:
          fName - First name of the salesperson
          lName - Last name of the salesperson
          comPercent - the commission % that the salesperson
          receives with each sale
     Outputs:
          None
     Return Value:
          None
     Salesperson(String fName, String lName, float comPercent);
     /*
     getSalespersonFName
          Gets the first name of the salesperson.
     Inputs:
          None
     Outputs:
          None
     Return Value:
          String fName - the salesperson's first name
     Exceptions:
          None */
     String getSalespersonFName();
     /*
     setSalespersonFName
          Sets the first name of the salesperson.
     Inputs:
          first - the new first name to be set
     Outputs:
          None
     Return Value:
          None
     Exceptions:
          None */
     void setSalesPersonFName(String first);
     /*
     getSalespersonLName
          Gets the last name of the salesperson.
     Inputs:
          None
     Outputs:
          None
     Return Value:
          String lName - the salesperson's last name
     Exceptions:
          None */
     String getSalespersonLName();
     /*
     setSalespersonLName
          Sets the first name of the salesperson.
     Inputs:
          last - the new last name to be set
     Outputs:
          None
     Return Value:
          None
     Exceptions:
          None */
     void setSalesPersonLName(String last);
     /*
     getComPercent
          Gets the commission percent of the salesperson.
     Inputs:
          None
```

```
Outputs:
     None
Return Value:
     float comPercent - the salesperson’s commission
     percent
Exceptions:
     None */
float getComPercent();
/*
setComPercent
     Sets commission percentage of the salesperson.
Inputs:
     cp - the new commission percentage to be set
Outputs:
     None
Return Value:
     None
Exceptions:
     None */
void setComPercent(float cp);
/*
getComTotal
     Gets the commission total of the salesperson.
Inputs:
     None
Outputs:
     None
Return Value:
     float comTotal - the salesperson’s commission total
Exceptions:
     None */
float getComTotal();
/*
setComTotal
     Sets commission total of the salesperson.
Inputs:
     ct - the new commission total to be set
Outputs:
     None
```

```
Return Value:
     None
Exceptions:
     None */
void setComTotal(float ct);
/*
getSalesTotal
     Gets the sales total of the salesperson.
Inputs:
     None
Outputs:
     None
Return Value:
     float salesTotal - the salesperson’s total sales
Exceptions:
     None */
float getSalesTotal();
/*
setSalesTotal
     Sets total sales of the salesperson.
Inputs:
     st - the new sales total to be set
Outputs:
     None
Return Value:
     None
Exceptions:
     None*/
void setSalesTotal(float st);
/*
getExistance
     Gets the existence of the salesperson.
Inputs:
     None
Outputs:
     None
Return Value:
     boolean exists - true if the salesperson exists and
     false if the salesperson has been “deleted”
```

```
Exceptions:
     none */
boolean getExistance();


/*
setSalesTotal
     Sets existence status of the salesperson.
Inputs:
     ex - the new existence to be set
Outputs:
     None
Return Value:
     None
Exceptions:
     None*/
void setExistence(boolean ex);
/*
getSalespersonName
     Gets the full name of a salesperson.
Inputs:
     None
Outputs:
     None
Return Value:
     String fullName - the full name of the salesperson
Exceptions:
     None */
String getSalespersonName();
/*
toString
     Displays all information about the salesperson in a
     string representation.
Inputs:
     None
Outputs:
     None
Return Value:
```

```
          String salespersonInfo - all of the salesperson’s
          information formatted in a readable way
     Exceptions:
          None */
     String toString();
}
```

SALESPERSON MANAGER

```
class SalespersonManager
{
     /* Class Description: The salesperson manager used to
     create, display, update (commission %), and delete a
     salesperson.*/
     /*
     salespersonManagerConstructor
          Creates the salesperson manager.
     Inputs:
          None
     Outputs:
          None
     Return Value:
          None
     Exceptions:
          None */
     SalespersonManager();
     /*
     salesMenu
          Displays the salesperson submenu and allows the user
          to select which option they would like to perform with
          salespeople.
     Inputs:
          None
     Outputs:
          None
     Return Value:
          None
     Exceptions:
```

```
     InvalidInput - if salesChoice is not one of the listed
     integers on the sales menu */
void salesMenu();
/*
salesCreate
     Creates a new Salesperson.
Inputs:
     None
Outputs:
     None
Return Value:
     None
Exceptions:
     DuplicateSalesperson - if the salesperson (identified
     by first and last name) already exists in the database
     InvalidCommission - if the commission % entered is
     less than 0% */
void salesCreate();
/*
salesUpdate
     Updates the commission percent of a salesperson.
Inputs:
     None
Outputs:
     None
Return Value:
     None
Exceptions:
     MissingSalesperson - if there are no salespeople
     existing in the database to be updated
     InvalidCommission - if the commission % entered is
     less than 0% */
void salesUpdate();
/*
salesDisplay
     Displays all existing salespeople.
Inputs:
     None
Outputs:
```

```
          None
     Return Value:
          None
     Exceptions:
          MissingSalesperson - if there are no salespeople
          existing in the database to be displayed */
     void salesDisplay();
     /*
     salesDelete
          Displays all existing salespeople and allows the user
          to select one to delete.
     Inputs:
          None
     Outputs:
          None
     Return Value:
          None
     Exceptions:
          MissingSalesperson - if there are no salespeople
          existing in the database to be deleted
          InvalidInput - if mainChoice is not one of the listed
          integers on the list*/
     void salesDelete();
}
```

PRODUCT

```
class Product
{
     /* Class Description: An individual product, contains its
     name, price and other relevant information.*/
     /*
     Product
          A constructor for the Product, allows the product to
          be instantiated with valid data.
     Inputs:
          name - the name of the product
          sellingPrice - the sales price of the product
          costPrice - the cost price of the product
          Quantity - The amount in inventory
     Outputs:
          None
     Return Value:
          None */
     Product(string name, float sellingPrice, float costPrice,
     int Quantity);
     /*
     setSalesPrice
          Modify the sales price of a product.
     Inputs:
          salesPrice - the sales price of the object, must be
          greater than or equal to 0.
     Outputs:
          None
     Return Value:
          None
     Exceptions:
          InvalidPrice - if the price is less than zero. */
     Signature:
          void setSalesPrice(float salesPrice);
     /*
     getSalePrice
          Get the sale price of a product.
     Inputs:
          none
     Outputs:
          None
     Return Value:
          float sellingPrice - the sale price of an object.
     Exceptions:
          none
     Signature:
          float getSalePrice();
     /*
     setCostPrice
          Modify the cost price of a product.
     Inputs:
          costPrice - the cost price of the object, must be
          greater than or equal to 0.
     Outputs:
          None
     Return Value:
          None
     Exceptions:
          InvalidPrice - if the price is less than zero. */
     Signature:
          void setCostPrice(float costPrice);
     /*
     getCostPrice
          Get the cost price of a product.
     Inputs:
          none
     Outputs:
          None
     Return Value:
          float costPrice - the cost price of an object.
     Exceptions:
          None */
     Signature:
          float getCostPrice();
     /*
     getAmountSold
          Displays the amount of the product sold. This variable
          is set to 0 at default on creation of the Product
          object.
     Inputs:
          None
     Outputs:
          None
     Return Value:
          int amountSold - indicates the amount of product sold
     Exceptions:
          None */
     int getAmountSold();
     /*
     getProfitTotal
```

```
     Get the difference between the sales price and the
     cost price.
Inputs:
     None
Outputs:
     None
Return Value:
     float profitTotal - The difference between the sales
     price and the cost price
Exceptions:
     None */
float getProfitTotal();
/*
getProfitPercent
     Returns the profit percentage for the product
Inputs:
     None
Outputs:
     None
Return Value:
     float profitPercentage - A float that contains the
     profit percentage.
Exceptions
     DivideByZero - this calculation involves division, if
     not initialized correctly this may result in divide by
     zero. */
float getProfitPercentage();
/*
getTotalSales
     Give the total sales amount of all of the products
     sold.
Inputs:
     None
Outputs:
     None
Return Value:
     float totalSales - A float that is the result of the
     sales quantity times the sales price.
Exceptions:
     None */
float getTotalSales();
/*
getTotalCost
     Gets the cost of all of the products, sold and on
     hand.
Inputs:
     None
Outputs:
     None
Return Value:
     float totalCost - the total cost of all products on
     hand and sold.
Exceptions:
     None */
float getTotalCost();
/*
adjustQuantity
     Increments the quantity by the specified amount, maybe
     be a negative amount to decrease the inventory. If
     successfully decreased it indicates a sale and the
     amount sold is incremented by that amount.
Inputs:
     amount - the amount to adjust the quantity by, maybe
     be a negative number.
Outputs:
     None
Return Value:
     None
Exceptions:
     InvalidAmount - if the amount causes the inventory to
     drop below zero, prevents amount sold from being
     incremented. */
void adjustQuantity(int amount);
/*
getQuantity
     Returns the amount of product in inventory, This
     variable is set to 0 by default upon creation of the
     object.
```

```
Inputs:
     None
Outputs:
     None
Return Value:
     int quantity - The amount of product on hand
Exceptions:
     None */
int getQuantity();
/*
toString:
     Displays all information about a product in a string
     representation.
Inputs:
     None
Outputs:
     None
Return Value:
     String productInfo - product name + sales price + cost
     price
Exceptions:
     None */
String toString();
/*
deleteProduct:
     Flags the product as removed from inventory, these
     products no longer show up in the list of available
     products, but can still be shown in invoices.
Inputs:
     None
Outputs:
     None
Return Value:
     None
Exceptions:
     None */
void deleteProduct();
/*
toString
```

```
          Displays all information about the product in a string
          representation.
     Inputs:
          None
     Outputs:
          None
     Return Value:
          String productInfo - all of the product’s information
          formatted in a readable way
     Exceptions:
          None */
     String toString();
}
```

PRODUCT MANAGER

```
class ProductManager
{
     /* Class Description: Contains the Product UI and Stores
     each product in a data structure.*/
     /*
     displayMenu
          Displays the product submenu and all of its submenus,
          collect user input for the required responses.
     Inputs:
          None
     Outputs:
          None
     Return Value:
          None
     Exceptions:
          InvalidInput - if productChoice is not one of the
          listed integers on the product menu */
     void displayProductMenu();

/*
searchForMatchingProduct
     Search the product list for a collision in the name of the
     product
Inputs:
     String productName - The name of the new product.
Outputs:
     None
Return Value:
     boolean productMatched - describes whether a match is found
     or not
Exceptions:
     None */
boolean searchForMatchingProduct(String productName);
/*
getProductList
     Returns product list
Inputs:
     None
Outputs:
     None
Return Value:
     ArrayList<Product> productList - returns the ArrayList of
     products.
Exceptions:
     None */
ArrayList<Product> getProductList();
}
```

INVENTORY

```
class Inventory
{
     /* Class Description: A class for products*/
     /*
     Inventory constructor
          makes a copy of the list from productManager
     Inputs:
          newProductList - a list to store products
     Outputs:
          None
     Return Value:
          None
```

```
Exceptions
     None */
Inventory(ProductManager newProductList);
/*
displayInventoryMenu
     Gives the user the option to select "1. display
     Inventory,"2. Display Inventory with 5 or fewer
     quantity", "3. Increase Inventory", or "4. return to
     main menu"
Inputs:
     None
Outputs:
     None
Return Value:
     None
Exceptions:
     invalidMenuOption - Occurs when user inputs an option
     that is not available */
void displayInventoryMenu();
/*
displayInventory
     Displays amount of products:  Product name, Selling
     Price, Cost Price, Quantity on Hand, Quantity Sold,
     Total Sales, Total Cost, Total Profit and Total Profit
     Percent.
Inputs:
     None
Outputs:
     None
Return Value:
     None
Exceptions:
     None */
void displayInventory();
/*
displayInventoryWith5orFewerQuantity
     System sorts and displays amount of on hand product
     (with 5 or less), Product name, Selling Price, Cost
```

```
          Price, Quantity Sold, Total Sales, Total Cost, Total
          Profit and Total Profit Percent
     Inputs:
          productList - list to get quantity from
     Outputs:
          None
     Return Value:
          None
     Exceptions:
          None */
     void displayInventoryWith5orFewerQuantity
     (ArrayList<Product> productList);
}
```

CUSTOMER

```
class Customer
{
     /* Class Description: A class storing customer data */
     /*
     Customer constructor
          creates a customer, with name, sales tax, address,
          make sure that the data is valid. Ie. numbers are
          numbers and names are strings.
     Inputs:
          name - Customers name
          salesTax -sales tax
          address - customers address.
     Outputs:
          None
     Return Value:
          None
     Exceptions:
          None */
     Customer(String name, float salesTax, String address);
     /*
     setName
          Modify the name of a Customer.
     Inputs:
```

```
     n - the new name to set for the customer.
Outputs:
     None
Return Value:
     None
Exceptions:
     None */
void setName(String n);
/*
getName
     Get the name of a Customer.
Inputs:
     None
Outputs:
     None
Return Value:
     String name - the name of the customer.
Exceptions:
     None */
String getName();

/*
setTax
     Modify the sales tax of a Customer.
Inputs:
     tax - the new sales tax to set for Customer.
Outputs:
     None
Return Value:
     None
Exceptions:
     None */
void setTax(float tax);
/*
getTax
     Get the sales tax of a Customer.
Inputs:
     None
Outputs:
     None
Return Value:
     float salesTax - the sales tax of the Customer.
Exceptions:
     None */
float getTax();
/*
setAddress
     Modify the address of a Customer.
Inputs:
     adrs - the new address to set for the Customer.
Outputs:
     None
Return Value:
     None
Exceptions:
     None */
void setAddress(String adrs);
/*
getAddress
     Get the address of a Customer.
Inputs:
     None
Outputs:
     None
Return Value:
     String address - the address of the Customer.
Exceptions:
     None */
String getAddress();
/*
toString
     Displays all information about the customer in a
     string representation.
Inputs:
     None
Outputs:
     None
Return Value:
```

```
          String customerInfo - all of the customer’s
          information formatted in a readable way
     Exceptions:
          None */
     String toString();
}
```

CUSTOMER MANAGER

```
class CustomerManager
{
     /* Class Description: A customer manager that creates,
     updates, and displays a list of Customers */
     /*
     CustomerManager constructor
          Creates the Customer Manager
     Inputs:
          None
     Outputs:
          None
     Return Value:
          None
     Exceptions:
          None */
     CustomerManager();
     /*
     customerCreate
          Creates a new Customer.
     Inputs:
          None
     Outputs:
          None
     Return Value:
          None
     Exceptions:
          DuplicateCustomer - if the customer (identified by
          first and last name) already exists in the database
          InvalidSalesTax - if the sales tax % entered is less
          than 0% */
```

```
void customerCreate();
/*
updateCustomerAddress
     A method to update a customer’s address
Inputs:
     None
Outputs:
     None
Return Value:
     None
Exceptions:
     AddressInvalid - Occurs when address is invalid. */
void updateCustomerAddress();
/*
updateCustomerSalesTax
     A method to update a customers sales tax
Inputs:
     None
Outputs:
     None
Return Value:
     None
Exceptions:
     salesTaxNotValid - Occurs when sales tax is less than
     zero or when an invalid character.*/
void updateCustomerSalesTax();
/*
displayCustomer
     A method to display a customer’s name, address, and
     sales tax.
Inputs:
     None
Outputs:
     None
Return Value:
     None
Exceptions:
     None */
void displayCustomer();
```

```
}

INVOICE

class Invoice
{
     /* Class Description: Object used for representing invoices
     which are made by the store owner and function as a request
     for payment. The invoice contains the following:
       1. Invoice number
       2. Date of Purchase
       3. Salesperson name
       4. Customer name, address
       5. Product(s): (name) - (qty) - (retail unit price))
       6. Sales Tax associated with Salesperson
       7. Freight charge (if method is shipping)
       8. Total $ amount of above.
       9. Total due (amount left to pay).
       10.  All payments made to an invoice.
            a. Displayed if Total_amount_due is less than
               total_invoice_amount */
     /*
     Invoice Constructor
          Constructor for empty Invoice object
     Inputs:
          None
     Outputs:
          None
     Return Value:
          None
     Exceptions:
          None */
     Invoice();
     /*
     Invoice Constructor Overloaded
          Overloaded Constructor for Invoice object.
     Inputs:
          int invoiceNum - identifying number of invoice.
          ArrayList <Product> productList - list of products
          Salesperson seller - object for salesperson
          Customer buyer - object for customer
          int[] dateOfPurchase - array for month, day, and year
     Outputs:
          None
     Return Value:
          None
     Exceptions:
          None */
     Invoice(int invoiceNum, ArrayList<Product> productList,
     Salesperson seller, Customer buyer, int[] dateOfPurchase);
     /*
     getInvoiceNumber
          Gets the associated invoice number
     Inputs:
          None
     Outputs:
          None
     Return Value:
          int invoice_num - associated invoice number
     Exceptions:
          None */
     int getInvoiceNumber();
     /*
     getProductList
          Gets the associated list of products
     Inputs:
          None
     Outputs:
          None
     Return Value:
          ArrayList<Product> productList - associated product
          list
     Exceptions:
          None */
     ArrayList<Product> getProductList();
     /*
     getSalesperson
          Gets the associated salesperson
```

```
Inputs:
     None
Outputs:
     None
Return Value:
     Salesperson seller - associated salesperson
Exceptions:
     None */
Salesperson getSalesperson();
/*
getCustomer
     Gets the associated customer
Inputs:
     None
Outputs:
     None
Return Value:
     Customer buyer - associated customer
Exceptions:
     None */
Customer getCustomer();
/*
getDateOfPurchase
     Gets the associated date of purchase
Inputs:
     None
Outputs:
     None
Return Value:
     Salesperson seller - associated date of purchase
Exceptions:
     None */
int[] getDateOfPurchase();
/*
toString
     Gets a string representation of Invoice
Inputs:
     None
Outputs:
```

```
          None
     Return Value:
          String invoiceInfo - all of the invoice’s information
          formatted in a readable way
     Exceptions:
          None */
     String toString();
}
```

<u>INVOICE MANAGER</u>

```
class InvoiceManager
{
     /* Class DescriptionObject used for representing invoice
     lists. */
     /*
     Invoice Manager Constructor
          Constructor for Invoice Manager object. Initializes an
          ArrayList of Invoices to store each new Invoice in.
     Inputs:
          None
     Outputs:
          None
     Return Value:
          None
     Exceptions:
          None */
     InvoiceManager();
     /*
     createInvoice
          Takes user input and creates new invoice via
          constructor call. When products are called in the
          creation of the invoice, The product quantity is
          updated (e.g. product decreases in qty). An integer
          array list holding the extensions corresponding to
          each product is generated, elements of which can be
          pulled to find profit in the product class.  Every
          invoice when created is assumed unpaid, and is
          represented as such by ‘total due’ variable equalling
          total amount of invoice. Once invoice is made, it is
          added to invoiceList
     Inputs:
          None
     Outputs:
          None
     Return Value:
          None
     Exceptions:
          None */
     void createInvoice();
     /*
     invoiceMenu_disp
          Displays options for invoice menu, which goes as
          follows:
               1. Display list of open invoices
               2. Display list of closed invoices
               3. Create a new invoice
               4. Display a specific invoice
               5. Pay an invoice
               6. Return to Main Menu
          Menu takes userInput and proceeds accordingly.
     Inputs:
          None
     Outputs:
          None
     Return Value:
          None
     Exceptions:
          InvalidSelection - if userInput is not within [1,6]*/
     void invoiceMenu_disp();
     /*
     open_invoiceList_disp
          Display all open invoices in invoiceList and total
          number of open invoices, and their payment(s). If
          total due > $0.00, then it is considered open. Option
          to return to main menu can be displayed. User can exit
          to main menu via enter number
     Inputs:
          None
     Outputs:
          None
     Return Value:
          None
     Exceptions:
          InvalidSelection - if userInput is not [1] (for
          exiting to main menu)*/
     void open_invoiceList_disp();
     /*
     closed_invoiceList_disp
          Display all closed invoices in invoiceList and total
          number of closed invoices and their payments. If total
          due = $0.00, then it is considered closed. Option to
          return to main menu can be displayed. User can exit to
          main menu via enter number.
     Inputs:
          None
     Outputs:
          None
     Return Value:
          None
     Exceptions:
          InvalidSelection - if userInput is not [1] (for
          exiting to main menu)*/
     void closed_invoiceList_disp();
     /*
     invoiceList_disp
          User is prompted to enter an invoice number to select
          invoice from invoiceList, after which the system
          displays the specific invoice and print its contents
          which include:
            1. Invoice number
            2. Invoice date_of_purchase
            3. Salesperson name
            4. Customer name, address
            5. Product(s): (name) - (qty) - (retail unit price)
               - (extension(s))
            6. Sales Tax associated with Salesperson
```

```
        7. Freight charge (if method is shipping)
        8. Total $ amount of above.
        9. Total due (amount left to pay).
            a. (IF TOTAL DUE IS LESS THAN TOTAL INVOICE $
               AMOUNT)
            b. All payments made to an invoice.
            c. If total due = 0, invoice is considered
               paid/closed. Once information is displayed,
               option to return to main menu can be
               displayed. User can exit to main menu via
               enter number
Inputs:
    None
Outputs:
    None
Return Value:
    None
Exceptions:
    InvalidSelection - if userInput is not in range of the
    length of the list. */
void invoice_disp();
/*
pay_invoice
    User selects invoice to pay from invoiceList via
    invoice_number. Once invoice is selected, System
    prompts for user to enter amount paid by customer.
    Once payment has been completed, system prompts for
    date of payment. User enters the date of payment.
    System compares amount to total due and the date of
    payment with date of purchase on invoice (if < 10
    days, total due = total due - discount) (if > 30 days,
    total due = total due + finance charge). A new object
    from <Payment> class is generated using the info
    given.
Inputs:
    None
Outputs:
    None
Return Value:
    None
Exceptions:
    InvalidSelection - if userInput is not in range of the
    length of the list. */
void pay_invoice();
/*
calculate_extension
    Multiply the product qty sold with the unit (retail)
    price. *Assuming ArrayList <Product> Product_list is
    already accessible.
Inputs:
    ArrayList<Float> qty_sold - amount of a product sold
    in an invoice
    ArrayList<Float> product_indices - selections of
    product from Product_list
Outputs:
    None
Return Value:
    float extension_for_product - float value holding
    total price of product (i.e. product qty * unit price)
Exceptions:
    None */
float calculate_extension(ArrayList<Float> qty_sold,
ArrayList<Float> product_indices);
/*
calculate_total
    Add the extensions, freight charge if applicable,
    sales_tax, and generate the invoice total $$
    amount(NOT THE TOTAL_AMOUNT_DUE)
Inputs:
    ArrayList<Float> product_extensions - integer array
    list holding extension values corresponding to a
    product array list.
Outputs:
    None
Return Value:
    int invoice_total - integer value holding total price
    of invoice.
Exceptions:
    None */
int calculate_total(arrayList<integer> product_extensions);
/*
calculate_discount
    Utilizes Class <Payment>. Compare date of invoice
    purchase and date of payment. If difference between
    purchase and payment date is less than 10, than
    discount generated is 10% of total_invoice_amount.  If
    the difference is greater than 10, then the discount
    is not generated.
Inputs:
    int[] date_of_purchase - date of invoice purchase to
    be compared to date of payment.
Outputs:
    None
Return Value:
    int discount - amount to be subtracted from
    total_left_due if full payment has been made within 10
    days of purchase.
Exceptions:
    None */
int calculate_discount(int[] date_of_purchase);>
/*
calculate_finance_charge
    Utilizes Class <Payment>. Compare date of invoice
    purchase and date of payment. If difference between
    purchase and payment date is greater than 30, than
    finance charge generated is 2% of
    total_invoice_amount. If difference is less than 30,
    finance charge is not generated and function is
    exited.
Inputs:
    int [] date_of_purchase - date of invoice purchase to
    be compared to date of payment.
Outputs:
    None
Return Value:
```

```
          integer finance_charge - amount to be added to
          total_left_due if full payment has been made past 30
          days of purchase.
     Exceptions:
          None
     int calculate_finance_charge(int[] date_of_purchase);
}
```

PAYMENT

```
class Payment
{
     /* Class representing a singular payment made to an
     invoice. A payment object is composed of:
       1. Invoice number - corresponds to invoice being paid.
       2. Pay date - date of payment by month, day, and year
       3. Payment amount - amount of $$$ being applied to total
          amount due in invoice.*/
     /*
     Payment Constructor
          Constructor for empty Invoice Payment object>
     Inputs:
          None
     Outputs:
          None
     Return Value:
          None
     Exceptions:
          None */
     Payment();
     /*
     Payment Constructor Overloaded
          Overloaded Constructor for Payment object
     Inputs:
          int invoice_num - corresponds to invoice being paid.
          float payment_amount - amount of $$$ being applied to
          total amount due in invoice.
          int[] pay_date - date of payment by month, day, and
          year
```

```
Outputs:
     None
Return Value:
     None
Exceptions:
     None */
Payment(int invoice_num, float payment_amount, int[]
pay_date);
/*
getInvoiceNumber
     Gets the associated invoice number
Inputs:
     None
Outputs:
     None
Return Value:
     int invoice_num - associated invoice number
Exceptions:
     None */
int getInvoiceNumber();
/*
getPaymentAmount
     Gets the associated payment
Inputs:
     None
Outputs:
     None
Return Value:
     float payment_amount - associated payment
Exceptions:
     None */
float getPaymentAmount;
/*
getPayDate
     Gets the associated pay date
Inputs:
     None
Outputs:
     None
```

```
    Return Value:
         int[] pay_date - associated invoice number
    Exceptions:
         None */
    int[] getPayDate();
    /*
    toString
         Gets a string representation of Payment
    Inputs:
         None
    Outputs:
         None
    Return Value:
         String paymentInfo - all of the payment’s information
         formatted in a readable way
    Exceptions:
         None */
    String toString();
}
```

PAYMENT MANAGER

```
class PaymentManager
{
    /* Class Description: Class representing list holding all
    payments made during a session. List starts off empty and
    is populated when pay_invoice() function is called, which
    generates a payment that is stored in the list.
    This class also includes the get_payment call needed when
    displaying all payments made to an invoice. */
    /*
    PaymentManager Constructor
         Constructor for empty Invoice Payment List object
    Inputs:
         None
    Outputs:
         None
    Return Value:
         None
    Exceptions:
         None */
    PaymentManager();
    /*
    payment_disp
         Displays all payments made to an invoice. (Called when
         displaying an invoice, IF number of total_amount_due
         DOES NOT equal total invoice amount).
    Inputs:
         int invoice_num - corresponds to invoice being paid.
    Outputs:
         None
    Return Value:
         None
    Exceptions:
         None */
    void payment_disp(int invoice_num);
}
```