```java
public class BhaktaBonnerScarsella003PA3
{
 public static void main(String[] args)
 {
   StockCostCalculator myClients;

   StockCostCalculator.start();

   System.exit(0);
 }  //END main
} //END APPLICATION class BhaktaBonnerScarsella003PA3
```

```java
/**
 * WARNING:  THE CODE IN THIS PROGRAM "CANNOT" BE ALTERED.
 * NO POSTING OF THIS CODE IS ALLOWED ANY WHERE AS IT
 * IS THE INTELLECTUAL PROPERTY OF THE AUTHOR.
 *
 * STUDENTS ARE TO INSERT CODE AND COMMENTS WHERE INDICATED IN CAPS
 * STARTING WITH "STUDENT INSERTS ... " OR CODE ... ".  USE DRJAVA'S
 * Find TO LOCATE THE INSERT AND CODE INSTRUCTIONS. *
 *
 * @(#)StockCost.java
 * @author Linda Shepherd
 * @version 1.00 2023/10/29 3:12 AM
 *
 * PROGRAM PURPOSE:  STUDENT INSERTS
 */
import java.util.Scanner;  //STUDENT INSERTS

public class StockCost
{
  //STUDENT INSERTS LINE COMMENTS FOR EACH FIELD
  private Scanner input = new Scanner(System.in);
  private String customerName;
  private String stockCostReport;

  /* NOTE:  EXCEPT FOR CONSTRUCTORS THAT DON'T HAVE A return TYPE,
   * METHODS THAT ARE INSTANCE METHODS REQUIRE AN OBJECT OF THE
   * CLASS TO CALL THEM IN THE CLIENT CLASS WHEREAS static METHODS
   * CAN BE CALLED USING THE NAME OF THE CLASS.
   */

  /**
   * STUDENT INSERTS DESCRIPTION OF WHAT'S GOING ON WITH THE CODE
   * INSIDE THE METHOD.
   */
  public StockCost()
  {
  }//END default constructor

  /**
   * STUDENT INSERTS DESCRIPTION OF WHAT'S GOING ON WITH THE CODE
   * INSIDE THE METHOD.
   */
  public StockCost(String customerName)//CODE THE CONSTRUCTOR'S HEADER BASED ON
THE CLOSE BRACE LINE COMMENT.
  {
    this.customerName = customerName;  //CODE THE ASSIGNMENT STATEMENT.

  }//END StockCost(customerName:  String)

  /**
   * STUDENT INSERTS DESCRIPTION OF WHAT'S GOING ON WITH THE CODE
   * INSIDE THE METHOD.
   */
  public StockCost(StockCost aStockCost)//CODE A COPY CONSTRUCTOR'S HEADER BASED
ON THE CLOSE BRACE LINE COMMENT.
  {
    this.aStockCost = aStockCost;  //CODE ASSIGNMENT STATEMENT FOR THE NAME
FIELD.

  }//END StockCost(aStockCost:  StockCost)

  /**
   * STUDENT INSERTS DESCRIPTION OF WHAT'S GOING ON WITH THE CODE
   * INSIDE THE METHOD.
   */
 public StockCost copy()//CODE THE METHOD HEADER BASED ON THE CLOSE BRACE LINE
COMMENT.
  {
    StockCost clone = new StockCost();  //CODE THE CREATION AND RETURN OF A
StockCost OBJECT CALLED clone.
    return clone;

  }//END copy():  StockCost

  /**
   * STUDENT INSERTS DESCRIPTION OF WHAT'S GOING ON WITH THE CODE
   * INSIDE THE METHOD.
```

```java
   */
  public void setCustomer(String ordinalSuffix)//CODE THE METHOD HEADER BASED ON
THE CLOSE BRACE LINE COMMENT.
  {
    String nameCopy = "";  //STUDENT INSERTS LINE COMMENT
    char correct = ' ';    //STUDENT INSERTS LINE COMMENT

    do
    {
      System.out.printf("%nEnter the name of the %s client:  ", ordinalSuffix);
      customerName = input.nextLine();

      /*Takes out all spaces in customerName and stores in the copy variable.*/
      nameCopy = new String(customerName).replace(" ", "");

      /*Resolves a customer's name that is not an alpha using isAlpha().*/
      while(!isAlpha(nameCopy))
      {
        System.out.printf("%nInvalid!  %s not alphabetic.  Please re-enter:  ",
customerName);
        customerName = input.nextLine();

        nameCopy = new String(customerName).replace(" ", "");
      }//while customer's name is NOT alphabetic

      customerName  = capitalize(customerName);

      System.out.printf("%nYou entered %s.  Is this correct?  \'Y\' or \'N\':  ",
customerName);
      correct = input.nextLine().toUpperCase().charAt(0);
    }while(correct != 'Y');  //do-while to validate customerName.

  }//END setCustomer(ordinalSuffix:  String):  void

  /**
   * STUDENT INSERTS DESCRIPTION OF WHAT'S GOING ON WITH THE CODE
   * INSIDE THE METHOD.
   */
  public int setShares()//CODE THE METHOD HEADER BASED ON THE CLOSE BRACE LINE
COMMENT.
  {
    System.out.printf("%nHow many shares do you want to purchase?  ");

    return validateInteger(input.hasNextInt());  //Returns the number of shares
after its data type is validated.
```

```
  }//END setShares():  int

  /**
   * STUDENT INSERTS DESCRIPTION OF WHAT'S GOING ON WITH THE CODE
   * INSIDE THE METHOD.
   */
  public double setSharePrice()//CODE THE METHOD HEADER BASED ON THE CLOSE BRACE
LINE COMMENT.
  {
    System.out.printf("%nWhat is the price per share?  ");

    return validateDouble(input.hasNextDouble());  //Returns the share price
after its data type is validated.

  }//END setSharePrice():  double

  /**
   * STUDENT INSERTS DESCRIPTION OF WHAT'S GOING ON WITH THE CODE
   * INSIDE THE METHOD.
   */
  public double setOnlineFee()//CODE THE METHOD HEADER BASED ON THE CLOSE BRACE
LINE COMMENT.
  {
    System.out.printf("%nWhat is the online fee?  ");

    return validateDouble(input.hasNextDouble());  //Return the online fee after
its data type is validated.

  }//END setOnlineFee():  double

  /**
   * STUDENT INSERTS DESCRIPTION OF WHAT'S GOING ON WITH THE CODE
   * INSIDE THE METHOD.
   */
  public double setCommissionRate()//CODE THE METHOD HEADER BASED ON THE CLOSE
BRACE LINE COMMENT.
  {
    System.out.printf("%nSet the commission rate as a decimal,
example:  0.02:  ");

    return validateDouble(input.hasNextDouble());  //Returns the commission rate
after its data type is validated.

  }//END setCommissionRate():  double
  /**
   * STUDENT INSERTS DESCRIPTION OF WHAT'S GOING ON WITH THE CODE
   * INSIDE THE METHOD.
   */
  public void storeStockCostRpt(String stockCostRpt)//CODE THE METHOD HEADER
BASED ON THE CLOSE BRACE LINE COMMENT.
  {
    stockCostReport = stockCostRpt;

  }//END storeStockCostRpt(stockCostRpt:  String)

  /**
   * STUDENT INSERTS DESCRIPTION OF WHAT'S GOING ON WITH THE CODE
   * INSIDE THE METHOD.
   */
  public String getCustomerName()//CODE THE METHOD HEADER BASED ON THE CLOSE
BRACE LINE COMMENT.
  {
    return customerName;  //CODE THE RETURN STATEMENT

  }//END getCustomerName():  String

  /**
   * STUDENT INSERTS DESCRIPTION OF WHAT'S GOING ON WITH THE CODE
   * INSIDE THE METHOD.
   */
  public String getStockCostRpt()//CODE THE METHOD HEADER BASED ON THE CLOSE
BRACE LINE COMMENT.
  {
    return stockCostRpt;  //CODE THE RETURN STATEMENT

  }//END getStockCostReport():  String

  /**
   * RECODED:  If an entry is not a valid integer, reprompts for a
   * valid integer continues until one is entered and returned.
   */
  public final int validateInteger(boolean validInteger)
  {
    int integerVal = 0;

    while(!validInteger)
    {
      input.next();
      System.out.printf("%nNot an integer!  Enter a valid integer:  ");
      validInteger = input.hasNextInt();
    }//END while NOT an integer

    integerVal = input.nextInt();

    input.nextLine();  //CLEAR BUFFER ACCOUNTED FOR

    return integerVal;

  }//END validateInteger(inputValue:  int):  static final int

  /**
   * RECODED:  If an entry is not a valid floating-point, reprompts
   * for a valid float continues until one is entered and returned.
   */
  public final double validateDouble(boolean validDouble)
  {
    double doubleVal = 0.0;

    while(!validDouble)
    {
      input.next();
      System.out.printf("%nNot a floating-point!  Enter a valid float:  ");
      validDouble = input.hasNextDouble();

    }//END while NOT a double

    doubleVal = input.nextDouble();

    input.nextLine();  //CLEAR BUFFER ACCOUNTED FOR

    return doubleVal;

  }//END validateDouble(validDouble:  boolean):  static final double

  /**
   * Tests whether a value is an alpha.
   */
  public static final boolean isAlpha(String word)
  {
    /* Strip of characters commonly found in names. */
    word = new String(word).replace(".", "");
    word = new String(word).replace(",", "");
```

```java
  /* Test to see if the word is not empty AND if each letter
   * in a word is an alphabetic character.
   */
  return word != null && word.chars().allMatch(Character :: isLetter);
 }//END isAlpha(word:  String):  static final boolean

 /**
  * RECODED:  Test to see if the incoming string is not empty AND
  * if each letter in the string is an alphabetic character.
  */
 public static final String capitalize(String str)
 {
  boolean found = false;  //Variable to determine if a dash is in the string.

  if(str.indexOf("-") >= 0)  //Does the dash exist?
  {
   found = true;  //The dash does exist.
   str = str.replace("-", " ");  //Replace the first occurence of the character
with a space.
  }//END if there is a dash

  String words[] = str.split("\\s");  //Each word in str is an element in the
array.
  String capitalized = "",  //Stores what came in the str with correct
capitalization.
   firstWord = "",  //Stores 1st letter of the str.
   wordAfter = "";  //Stores the remaining letters in the str.

  for(String aWord : words)
  {
    firstWord = aWord.substring(0, 1);  //Get the first character.
    wordAfter = aWord.substring(1);     //Get the rest of the characters
starting at the 2nd.
    capitalized += firstWord.toUpperCase() + wordAfter.toLowerCase() + "
";  //Join capitalized words.
  }//for each word from a String in the words array, capitalize the first
letter

  if(found)  //If there was a dash, put it back in.
  {
   capitalized = capitalized.replaceFirst(" ", "-");  //Put dash back into
first blank space.
  }//if found
```

```
    return capitalized.trim();  //Return the string with the first letters all
capitalized.
  }//END capitalize(str:  String):  static final String

}//END CLASS StockCost
```

```
import java.util.Scanner;
import java.util.Calendar;
import java.io.File;
import java.io.PrintWriter;
import java.io.IOException;
//import my.package.StockCost;
public class StockCostCalculator
{


  private Scanner input = new Scanner(System.in);
  private StockCost[] stockCostCalcs;
  private String brokerageFirm;
  private String fileName;
  private char correct;
  private char another;

  public start()
  {
    System.out.printf("%nBegin entering for stock cost calculations? \'Y\' or
\'N\':  ");

    another = input.nextChar().toUpperCase();

    while(!Character.isLetter(another) || (another != 'Y' && another != 'N'))
    {
      System.out.printf("%nYou entered %s which is not a letter or not a Y or N
                          + "for your response to begin entering for stock
calculations.
                          + "%n%nPlease re-enter \'Y\' or \'N\':  ", another);

      another = input.nextChar().toUpperCase();

      if(another = 'Y')
      {
        createStockCostReport();
        writeStockCostReport();
        printStockCostReports();
      }  // END if(another = 'Y')
      else
      {
        System.out.printf("%nExiting program.%n");
      }
    } //END while(!Character.isLetter(another) || (another != 'Y' && another !=
'N'))


  }  //END start()

  public void createStockCostReport()
  {
    int noClients = 0;
    int shares = 0;
    int noStocks = 0;

    String ordinalSuffix = "";

    char anotherStock = '';
    char onlineTrade = '';
    char brokerAssisted = '';

    double stockCost = 0.0;
    double commission = 0.0;
    double totalCost = 0.0;
    double onlineFee = 0.0;
    double totalCost = 0.0;
    double totalCommissions = 0.0;
    double totalOnlineFees = 0.0;
    double sharePrice = 0.0;
    double commissionRate = 0.0;

    setBrokerageFirm();

    System.out.printf("%nYou\'ll be generating stock cost calculations for how
many clients?%n");

    while(!input.hasNextInt())
    {
      input.next();

      System.out.printf("%nInvalid integer! Re-enter the number of clients:  ");
      noClients = input.nextInt();
      input.next();

      stockCostCalcs[noClients];

      for(int i = 0; i < noClients; i++)
      {
        noStocks = 0;
        totalCommissions = 0.0;
        totalOnlineFees = 0.0;
        totalStockCost = 0.0;
        totalCost = 0.0;

        switch((i + 1) % 10)
        {
          case 1:
            ordinalSuffix = "st";
            break;
          case 2:
            ordinalSuffix = "nd";
            break;
          case 3:
            ordinalSuffix = "rd";
            break;
          default:
            ordinalSuffix = "th";
        }//END switch((i + 1) % 10)

        ordinalSuffix = String.valueOf(i + 1) + ordinalSuffix;

        stockCostCalcs[i] = new StockCost();

        stockCostCalcs[i].setCustomerName(ordinalSuffix);

        System.out.printf("%Enter \'Y\' to begin stock cost calculations or \'N\'
to exit:  ");
        anotherStock = input.nextChar().toUpperCase();

        while(!Character.isLetter(anotherStock) || (anotherStock != 'Y' &&
anotherStock != 'N'))
        {
          System.out.printf("%nYou entered %s which is not a letter or not a Y or
N for your response to
                              + "begin entering for stock cost calculations.
                              + "%n%nPlease re-enter \'Y\' or \'N\':  ",
anotherStock)
```

```
            anotherStock = input.nextChar().toUpperCase();
        }  //END while(!Character.isLetter(anotherStock) || (anotherStock != 'Y'
&& anotherStock != 'N'))

        while(anotherStock == 'Y')
        {
          ++noStocks;

          shares = stockCostCalcs[i].setShares();
          sharePrice = stockCostCalcs[i].setSharePrice();

          stockCost = shares * sharePrice;
          totalStockCost += stockCost;
          totalCost += stockCost;

          if(anotherStock == 'Y')
          {
            promptOnlineTrade();

            onlineFee = stockCostCalcs[i].setOnlineFee();

            totalOnlineFees += onlineFee;
            totalCost += onlineFee;
          }  //END if(anotherStock == 'Y')
          else
          {
            System.out.printf("%nINVALID TRADE TYPE!%n");

            --noStocks;

            totalStockCost -= stockCost;
            totalCost -= stockCost;
          }  //END else

          System.out.printf("%Enter \'Y\' to continue with another stock
calculation or \'N\' to exit:  ");
          anotherStock = input.nextChar.toUpperCase();

          while(!Character.isLetter(anotherStock) || (anotherStock != 'Y' &&
anotherStock != 'N'))
          {
            System.out.printf("%nYou entered %s which is not a letter or not a Y
or N for your response to
                              + "begin entering for stock cost calculations.
                              + "%n%nPlease re-enter \'Y\' or \'N\':  ",
anotherStock);
            anotherStock = input.nextChar().toUpperCase();
          }//END while(!Character.isLetter(anotherStock) || (anotherStock != 'Y'
&& anotherStock != 'N'))

          if(noStocks > 0 && anotherStock == 'N')
          {
            String stockCostRpt = String.format("%nSTOCK COST REPORT%n");
            stockCostRpt += formatFinalOutput(getCustomerName(i), totalStockCost,
totalOnlineFees, totalCommissions, totalCost);
          } //END if(noStocks > 0 && anotherStock == 'N')
          else
          {
            stockCostRpt = "";
          } //END if !(noStocks > 0 && anotherStock == 'N')
        } //END while(anotherStock == 'Y')

        if(stockCostRpt != null)
        {
          stockCostCalcs[i] = stockCostRpt;
          stockCostReport(stockCostRpt);
        } //END if(stockCostRpt != null)
      }  //END for(int i = 0; i < noClients; i++)
    }  //END while(!input.hasNextInt())
  }  //END createStockCostReport

  public void setBrokerageFirm()
  {
    do
    {
      System.out.printf("%nEnter the name of the brokerage firm:  ");
      brokerageFirm = StockCost.capitalize(input.nextLine());

      System.out.printf("%nYou entered %s. Is this correct? \'Y\' or \'N\':  ",
brokerageFirm);

      while(!Character.isLetter(correct) || (correct != 'Y' && correct != 'N'))
      {
        correct = input.nextChar().toUpperCase();
        System.out.printf("%nYou entered %s which is not a letter or not a Y or N
for your response to:  "
                                     + "%n%nYou entered %s. Is this correct?
Please re-enter \'Y\' or \'N\':  ", correct, brokerageFirm);
```

```
      } //END while(!Character.isLetter(anotherStock) || (anotherStock != 'Y' &&
anotherStock != 'N'))
    } while (correct == 'N')
  }  //END setBrokerageFirm : void

  public char promptOnlineTrade()
  {
    char onlineTrade = '';

    System.out.printf("%nIs this an online trade? Enter \'Y\' or \'N\':  ");
    onlineTrade = input.nextChar().toUpperCase();

    while(!Character.isLetter(onlineTrade) || (onlineTrade != 'Y' && onlineTrade
!= 'N'))
    {
      System.out.printf("%nYou entered %s which is not a letter or not a Y or N
for your response to:  "
                                  + "%n%nIs this an online trade? Please re-enter
\'Y\' or \'N\':  ", onlineTrade);
      onlineTrade = input.nextChar().toUpperCase();
    } //END while(!Character.isLetter(onlineTrade) || (onlineTrade != 'Y' &&
onlineTrade != 'N'))

    return onlineTrade;

  }  //END promptOnlineTrade: void

  public char promptBrokerAssisted()
  {
    char brokerAssisted = '';

    System.out.printf("%nIs this a broker assisted trade? Enter \'Y\' or
\'N\':  ");
    brokerAssisted = input.nextChar().toUpperCase();

     while(!Character.isLetter(brokerAssisted) || (brokerAssisted != 'Y' &&
brokerAssisted != 'N'))
     {
       System.out.printf("%nYou entered %s which is not a letter or not a Y or N
for your response to:  "
                                  + "%n%nIs this a broker assisted trade? Please
re-enter \'Y\' or \'N\':  ", brokerAssisted);
       brokerAssisted = input.nextChar().toUpperCase();
     } //END while(!Character.isLetter(brokerAssisted) || (brokerAssisted != 'Y'
&& brokerAssisted != 'N'))
```

```java
     return brokerAssisted;

  }  //END promptBrokerAssisted

  public static String formatFinalOutput(String customerName, double
totalStockCost, double totalOnlineFees, double totalCommissions, double
totalCost)
  {
    Calendar dateTime = Calendar.getInstance();
    String date = "";
    date = String.format("%1$TB %1$Td, %1$TY", dateTime);

    return String.format("%nYEE-TRADE, INC."
                         + "%nTOTAL COST OF INTENDED STOCK PURCHASES "
                         + "%nFOR %s"+ "%nAS OF %s"
    //3 spaces before the format specifiers through commissions.
                         + "%n%nTotal Stock Cost:   $%,14.2f"
                         + "%nTotal Online Fees:   %14s"
                         + "%nTotal Commissions:   %14s"
    //9 spaces before the format specifier for TOTAL COST.
                         + "%n%nTOTAL COST:         $%,14.2f%n", customerName,
                       date, totalStockCost, String.format("%,.2f",
                       totalOnlineFees), String.format("%,.2f",
totalCommissions),
                       totalCost);

  }  //END formatFinalOutput

  public void writeStockCostReports()
  {
    String stockCostReport = "";

    PrintWriter outputFile;

    boolean fileError;

    try
    {
      System.out.printf("%nEnter the file name for the stock cost reports with
the �txt�extension."
                          + "%n(WARNING: This will erase a pre-existing
file!):  ");
      fileName = input.nextLine();
      PrintWriter outputFile = new PrintWriter(fileName);

      for(int i = 0; i < stockCostCalcs.length; i++)
      {
        String stockCostReport = String.format("%s%n",
getStockCostReport(stockCostCalcs[i]));
        outputFile.printf("%s", stockCostReport);
      } // END for(int i = 0; i < stockCostCalcs.length; i++)
    } //END try
    catch(IOException e)
    {
      System.err.printf("%nFile cannot be created.");
      fileError = true;
    } //END catch

    if(!fileError)
    {
     outputFile.close();

     System.out.printf("%nData written to %s file.", fileName);
    } //END if(!fileError)

  }  //END writeStockCostReports

  public void printStockCostReports()
  {

  }  //END printStockCostReports


}  //END Application Class
```