5 Pages

JAVA PROGRAMMING (45)

Technical Task

KEY

Regional – 2013

Judges/Graders:

Please double-check and verify all scores!

Java Programming

Technical Task Scoring Sheet:

Application/Execution

- Application reads from the Orders.txt file …… _____ 30 pts
- Applications correctly writes to the OrdersProcessed.txt file … _____ 40 pts
- Application correctly calculates tax, shipping and total order cost (*3@15 ea)* …… _____ 45 pts
- Application displays Start/Finished messages to the screen … _____ 10 pts
- Application reads the entire file ……... _____ 30 pts

Tax and Shipping constants are defined …….. _____ 10 pts
OrderProcessor Class is implemented ……. _____ 50 pts
OrderProcessor Class has appropriate constructor(s) …….. _____ 20 pts
OrderProcessor Class has method to open input/output files …….. _____ 20 pts
OrderProcessor Class has method to close input/output files ……. _____ 10 pts
OrderProcessor Class has method to read/process input orders …….. _____ 50 pts
OrderProcessor Class has method to calculate tax, shipping, and write orders processed data to the output file …… _____ 50 pts
I/O error handling is done if files cannot be opened …… _____ 10 pts
JavaDoc comments are used …… _____ 10 pts
Code is modularized …… _____ 20 pts
Methods are commented ……. _____ 10 pts
OrderProcessor class is properly commented ……. _____ 10 pts
Code copied to USB drive and program runs from USB ……. _____ 10 pts

Total Points: _____ 435 pts

Total Points: _____ 510 pts

Note to Grader: If you would like to run the java program, place the input text file in the root of the application where it can be located by the application. This program does run in Eclipse as written

Sample Solution code. Contestant code may vary. The class code is given first.

```
/**
 * @author team name
 *
 * This class will open an order input file and order output file provided by the
program caller.  It will read each line and parse each
 * element based on the string token "|".  Contained in the input file is the
 * ORDER_ID|PART_NUMBER|PRICE|QUANTITY
 * The elements are passed to a method that will calculate tax(2%), shipping (5%),
and total order cost and write all the info to the
 * designated output file.
 */
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.PrintWriter;
import java.util.Scanner;
import java.util.StringTokenizer;

public class OrderProcessor {

	/**
	 * Instance variables
	 */
	private Scanner inFile = null;
	private PrintWriter outFile = null;

	/**
	 * Instance constants
	 */
	private static final double TAX_RATE = 0.02;
	private static final double SHIP_RATE = 0.05;

	/**
	 * Default constructor.  Opens the input and output files with default files.
	 */
	public OrderProcessor() {
		openFiles("hard coded default input file path", "hard coded default
input file path");
	}

	/**
	 * constructor.  Opens input and output files with values passed in by running
job.
	 * @param inFilePath
	 * @param outFilePath
	 */
	public OrderProcessor(String inFilePath, String outFilePath) {
		this.openFiles(inFilePath,outFilePath);
	}
```

```
	/**
	 * Opens the input and output files and logs an error if one cannot be opened.
	 *
	 * @param inFilePath
	 * @param outFilePath
	 */
	private void openFiles(String inFilePath, String outFilePath) {
		try {
			inFile = new Scanner (new FileReader(inFilePath));
			outFile = new PrintWriter(outFilePath);
		} catch (FileNotFoundException e) {
			System.out.println("Unable to open input/output order file, " +
inFilePath + ", " + outFilePath);
			System.out.println(e.getMessage());
		}
	}

	/**
	 * Closes the input and output files.
	 */
	private void closeFiles() {
		inFile.close();
		outFile.close();
	}

	/**
	 * This method reads each line of the input file, parses each element of the
input line and pass them to the writeOutput method.
	 * It makes a call to the closeFiles method to close both files when finished.
	 */
	public void processOrders() {
		String nextLine;
		StringTokenizer st;
		String orderId;
		String partNum;
		Double price;
		Integer quantity;

		nextLine = inFile.nextLine();
		while(inFile.hasNext()) {
			nextLine = inFile.nextLine();
			st = new StringTokenizer(nextLine, "|");
			orderId = st.nextToken();
			partNum = st.nextToken();
			price = new Double(st.nextToken());
			quantity = new Integer(st.nextToken());
			writeOutput(orderId, partNum, price, quantity);
		}

		closeFiles();
	}

	/**
```

```
        * Method that prints the order information to the output file.  It adds tax,
shipping, and total order cost
        * as additional information.
        *
        * @param orderId
        * @param partNum
        * @param price
        * @param quantity
        */
       private void writeOutput(String orderId, String partNum, Double price, Integer
quantity) {
              double itemAmount = price.doubleValue() * quantity.doubleValue();
              double taxAmount  = itemAmount * TAX_RATE;
              double shipAmount = itemAmount * SHIP_RATE;
              double totalAmount = itemAmount + taxAmount + shipAmount;

              outFile.write("Order Id: \t" + orderId + "\n");
              outFile.write("Part Num: \t" + partNum + "\n");
              outFile.write("Price:    \t" + price.toString() + "\n");
              outFile.write("Quantity: \t" + quantity.toString() + "\n");
              outFile.write("Tax:      \t" + taxAmount + "\n");
              outFile.write("Ship:     \t" + shipAmount + "\n");
              outFile.write("Total:    \t" + totalAmount + "\n");
              outFile.write("\n\n");
              outFile.flush();
       }
}
```

Main program:

```
/**
 * @author team name
 *
 * This class is used to test the OrderProcessor class.
 */
public class TestOrderProcessor {
       /**
        * The main method to start the program from the command line.
        * @param args
        */
       public static void main(String[] args) {
              OrderProcessor orderProcessor = new OrderProcessor("Orders.txt",
"OrdersProcessed.txt");

              System.out.println("Start processing orders.");
              orderProcessor.processOrders();
              System.out.println("Finished processing orders.");
       }
}
```

Note to Grader: When running the application, use the *grader_Orders.txt* file provided. Make sure you rename it to *Orders.txt* first. Since no dollar formatting is being done the extended

decimal values may vary so focus on the at most three digits to the right of the decimal point. The following is the correct output:

```
Orders:
ORDER_ID|PART_NUM|PRICE|QUANTITY
1|1|1|1
9872|MFT_903|4.99|1
2342|1900|3.50|4
9122|ACT_MIT|57.95|2
ORD123|112235|0.99|25
0982|FRM_102_221|99.99|1

Processed Orders:
Order Id:       1
Part Num:       1
Price:          1.0
Quantity:       1
Tax:            0.02
Ship:           0.05
Total:          1.07


Order Id:       9872
Part Num:       MFT_903
Price:          4.99
Quantity:       1
Tax:            0.0998
Ship:           0.24950000000000003
Total:          5.339300000000001


Order Id:       2342
Part Num:       1900
Price:          3.5
Quantity:       4
Tax:            0.28
Ship:           0.7000000000000001
Total:          14.979999999999999


Order Id:       9122
Part Num:       ACT_MIT
Price:          57.95
Quantity:       2
Tax:            2.318
Ship:           5.795000000000001
Total:          124.013
```

```
Order Id:       ORD123
Part Num:       112235
Price:          0.99
Quantity:       25
Tax:            0.495
Ship:           1.2375
Total:          26.4825


Order Id:       0982
Part Num:       FRM_102_221
Price:          99.99
Quantity:       1
Tax:            1.9998
Ship:           4.9995
Total:          106.98929999999999
```