

Name: Lum Chee Xiang Michael (Lan Zhixiang)
Student ID: 111102750
Assignment 3

Module: Introduction to Java and Object-oriented Programming
Module number: CO1109-01

University of London International Programmes
BSc in Computing and Informations: Introduction to Java and Object-oriented programming
Module number: CO1109-01
Assignment 3

Name: Lum Chee Xiang Michael (Lan Zhixiang)
Student ID: 111102750
Assignment 3

Module: Introduction to Java and Object-oriented Programming
Module number: CO1109-01

Source Codes

Working class: working.java

```
import java.util.*;
import java.io.*;
import java.text.*;

public class working
{
    product stuff[] = new product[9];
    DecimalFormat f = new DecimalFormat("0.00");
    double ttcost = 0;
    double shipcost = 0;
    int ttweight = 0;
    int units = 0;
    int check = 0;

    public static void main(String[] args)
    throws IOException
    {
        working a = new working();
        a.initialize();
    }

    public void playInventory() //this method is used to display the product catalogue when the program is run
    throws IOException
    {
        Scanner in = new Scanner(new FileReader("inventoryDisplay.txt"));
        while (in.hasNext())
        {
            for(int i=0; i < 3; i = i + 1)
            {
                System.out.print(in.nextLine());
            }
            System.out.println(in.nextLine());
        }
        in.close();
    }

    public void showInventory() //this method is used to display the product catalogue headers when the program is run
    {
        System.out.println("  HARDWARE ITEMS");
        System.out.println("CODE      DESCRIPTION                WEIGHT(g)  UNIT PRICE  ");
    }
}
```

Name: Lum Chee Xiang Michael (Lan Zhixiang)
Student ID: 111102750
Assignment 3

Module: Introduction to Java and Object-oriented Programming
Module number: CO1109-01

```
public void initialize() //this method starts the entire program by calling specific methods from the order class, where most  
of the processing is done  
    throws IOException  
    {  
        order o = new order();  
        showInventory();  
        playInventory();  
        System.out.println("");  
        o.readFile();  
        o.shipAndtax();  
        o.startOrder();  
    }  
}
```

Name: Lum Chee Xiang Michael (Lan Zhixiang)
Student ID: 111102750
Assignment 3

Module: Introduction to Java and Object-oriented Programming
Module number: CO1109-01

Order class: order.java

```
import java.util.*;
import java.io.*;
import java.text.*;
```

```
public class order
{
    Scanner in = new Scanner(System.in);
    product stuff[] = new product[9];
    DecimalFormat f = new DecimalFormat("0.00");
    double ttcost = 0; //this variable handles the total cost of the products before shipment and taxes are considered
    double shipcost = 0; //this variable handles the total shipment costs for the customer's order
    int tweight = 0; //this variable handles the total weight of the customer's order
    int units = 0; //this variable calculates the number of 250g blocks the customer's total purchase weight contains
    int check = 0; //this variable is used to check of the customer's total purchase weight is perfectly divisible by 250
    double VAT = 0; //actual value is eventually provided by FileReader in method shipAndtax()
    double ship = 0; //actual value is eventually provided by FileReader in method shipAndtax()
    customer a = new customer(); //this is where a new customer object is created to store the customer's billing details

    public void readFile() //this method reads in the various fields that makes up the product catalogue into individual object arrays from the specified file
        throws IOException
    {
        Scanner in = new Scanner(new FileReader("inventoryvalues.txt"));
        while (in.hasNext())
        {
            for(int i=0 ; i <= 8 ; i = i + 1)
            {
                stuff[i] = new product();
                stuff[i].pid=in.nextLine();
                stuff[i].pdesc=in.nextLine();
                stuff[i].weight= Integer.parseInt(in.nextLine());
                stuff[i].cost=Double.parseDouble(in.nextLine());
            }
        }
        in.close();
    }

    public void shipAndtax() //this method retrieves the shipping and VAT rates from a separate text file specified within
        throws IOException
    {
        Scanner abs = new Scanner(new FileReader("addoncosts.txt"));
        while (abs.hasNext())
        {
            VAT = Double.parseDouble(abs.nextLine());
            ship = Double.parseDouble(abs.nextLine());
        }
        abs.close();
    }
}
```

```
public void getCustomer() //this method calls the Customer class to prompt a customer to enter his billing details
{
```

```
    a.billinfo();
    a.displayCustInfo();
}
```

```
public void eraseOldOrders() //this method resets all previous purchases to zero for a new transaction to take place
{
```

```
    for (int r = 1; r <= 9; r = r + 1)
    {
        stuff[r-1].orderquantity = 0;
        stuff[r-1].totalweight = 0;
        stuff[r-1].totalcost = 0;
    }
}
```

```
public void takeXX() //this method specifies what the program should do when 'XX' is entered in the order-taking process
    throws IOException
```

```
{
    int tquantity = 0;
    for (int z = 1; z <= 9; z = z + 1)
    {
        tquantity = tquantity + stuff[z-1].orderquantity;
    }
    if (tquantity == 0)
    {
        takeNothing();
    }

    else
    {
        writeInvoice();
        takeInvoice();
    }
}
```

```
public void takeNothing() //this method tells the program what to print if the customer does not order anything
```

```
{
    System.out.println("You have ended your session without ordering anything.");
}
```

throws IOException

6

7

```
public void startOrder() //this method commences the order-taking process
    throws IOException
{
    boolean breaker = true;
    while (breaker == true)
    {

        eraseOldOrders();
        getCustomer();
        System.out.println("\nEnter CODE: (XX to Stop)");
        boolean success = false;
        while (true)
        {
            System.out.println("CODE: ");
            String input = in.next();
            if (input.equalsIgnoreCase("XX"))
            {
                takeXX();
                break;
            }
            for(int i = 0 ; i< stuff.length;i++)
            {
                if(input.equalsIgnoreCase(stuff[i].pid))
                { success = true;
                  stuff[i].getorderQuantity();}

                if(success==false && i==stuff.length - 1)
                {
                    System.out.println("You have entered an invalid code. Please try again.");
                }
            }
        }

        end n = new end();
        n.endcase();
        in.nextLine();
        while (true)
        {
            System.out.print("> ");
            String nextpls = in.nextLine();
            if (nextpls.equalsIgnoreCase("no") || nextpls.equalsIgnoreCase("n"))
            {
                n.endNo();
                breaker = false;
                break;
            }
            else if (nextpls.equalsIgnoreCase("yes") || nextpls.equalsIgnoreCase("y"))
            {
                n.endYes();
                break;
            }
        }
    }
}
```


Name: Lum Chee Xiang Michael (Lan Zhixiang)
Student ID: 111102750
Assignment 3

Module: Introduction to Java and Object-oriented Programming
Module number: CO1109-01

```
        }  
        else  
        {  
            System.out.println("\nInvalid response. Please try again.");  
        }  
    }  
}  
}
```

Name: Lum Chee Xiang Michael (Lan Zhixiang)
Student ID: 111102750
Assignment 3

Module: Introduction to Java and Object-oriented Programming
Module number: CO1109-01

Customer class: customer.java

```
import java.util.*;
import java.io.*;

public class customer
{
    public String fullname;
    public String saddress;
    public String bnumber;
    public String state;
    public String pcode;

    public customer()
    {

    }

    public String fullname() //this method prompts the customer to enter his full name
    {
        Scanner in = new Scanner(System.in);
        System.out.println("Please enter your full name: ");
        System.out.print("> ");
        fullname = in.nextLine();
        return fullname;
    }

    public String address1() //this method prompts the customer to enter his Street address
    {
        Scanner in = new Scanner(System.in);
        System.out.println("Please enter your Street address: ");
        System.out.print("> ");
        saddress = in.nextLine();
        return saddress;
    }

    public String address2() //this method prompts the customer to enter his House or Apartment number
    {
        Scanner in = new Scanner(System.in);
        System.out.println("Please enter your House or Apartment number:");
        System.out.print("> ");
        bnumber = in.nextLine();
        return bnumber;
    }
}
```

```
public String state() //this method prompts the customer to enter his state of residence
{
    Scanner in = new Scanner(System.in);
    System.out.println("Please enter your State of residence: ");
    System.out.print("> ");
    state = in.nextLine();
    return state;
}

public String pcode() //this method prompts the customer to enter his postal code
{
    Scanner in = new Scanner(System.in);
    System.out.println("Please enter your postal code: ");
    System.out.print("> ");
    pcode = in.nextLine();
    return pcode;
}

public void billinfo() //this method combines all the previous prompts into a single method
{
    fullname();
    address1();
    address2();
    state();
    pcode();
}

public void displayCustInfo() //this method displays the customer's billing information after it has been entered
{
    System.out.println("PLACE YOUR ORDER!");
    System.out.println("NAME: " + fullname);
    System.out.println("ADDRESS-1: " + saddress);
    System.out.println("ADDRESS-2: " + bnumber);
    System.out.println("ADDRESS-3: " + state);
    System.out.println("POST CODE: " + pcode);
}

public void invoiceCustomer() //this method displays the customer's billing information in the invoice
{
    System.out.println(fullname);
    System.out.println(saddress);
    System.out.println(bnumber);
    System.out.println(state + "    POST CODE: " + pcode);
    System.out.println();
}
```

Name: Lum Chee Xiang Michael (Lan Zhixiang)
Student ID: 111102750
Assignment 3

Module: Introduction to Java and Object-oriented Programming
Module number: CO1109-01

```
public String invoiceCustomerfile() //this method displays the customer's billing information in the text file of the invoice
{
    String detail = fullname + "\n" + saddress + "\n" + bnumber + "\n" + state + "        POST CODE: " + pcode;
    return detail;
}
}
```

Name: Lum Chee Xiang Michael (Lan Zhixiang)
Student ID: 111102750
Assignment 3

Module: Introduction to Java and Object-oriented Programming
Module number: CO1109-01

Product class: product.java

```
import java.util.*;
import java.io.*;
import java.text.*;

public class product
{
    public String pid; //this variable contains the various product IDs
    public String pdesc; //this variable contains the various product descriptions
    public int weight; //this variable contains information about the product's weight
    public double cost; //this variable contains information about the product's pricing
    public int orderquantity; //this variable contains information about the amount ordered for each product
    public double totalcost; //this variable contains information about the total cost for each product ordered by the customer
    public int totalweight; //this variable contains information about the total weight for each ordered product
    DecimalFormat f = new DecimalFormat("0.00"); //this forcibly expresses various numerical values in two decimal places

    public void getorderQuantity() //this method is used to check whether the product quantity has exceeded 100 or is negative, and to calculate the weight of the order
    {
        Scanner in = new Scanner(System.in);
        System.out.println("QUANTITY: ");
        orderquantity = orderquantity + in.nextInt();
        {
            while (orderquantity > 100 || orderquantity < 0)
            {
                System.out.println("Sorry, your total quantity for this product has exceeded 100 or is less than 0. Please re-enter the quantity again: ");
                System.out.println("QUANTITY: ");
                orderquantity = 0;
                orderquantity = orderquantity + in.nextInt();
            }
            totalweight = orderquantity * weight;
            totalcost = orderquantity * cost;
        }
    }
}
```

Name: Lum Chee Xiang Michael (Lan Zhixiang)
Student ID: 111102750
Assignment 3

Module: Introduction to Java and Object-oriented Programming
Module number: CO1109-01

Ending class: end.java

```
import java.util.*;
import java.io.*;

public class end
{
    public end()
    {
    }
    public boolean endNo() //this method is used to handle a case where there are no more transactions to be done
    {
        System.out.println("\nThank you for shopping with us. The program will now terminate.");
        return false;
    }

    public boolean endYes() //this method is used to handle a case where there are additional transactions to be done
    {
        return true;
    }

    public void endcase() //this method is used to determine if the program is still required to handle additional transactions
    {
        order o = new order();
        Scanner in = new Scanner(System.in);
        System.out.println("\nIs there another customer waiting to perform a transaction on this machine?");
        System.out.print("Enter N for no, or Y or yes, (followed by the 'Enter key') to proceed.");
    }
}
```

Name: Lum Chee Xiang Michael (Lan Zhixiang)
Student ID: 111102750
Assignment 3

Module: Introduction to Java and Object-oriented Programming
Module number: CO1109-01

External text files used

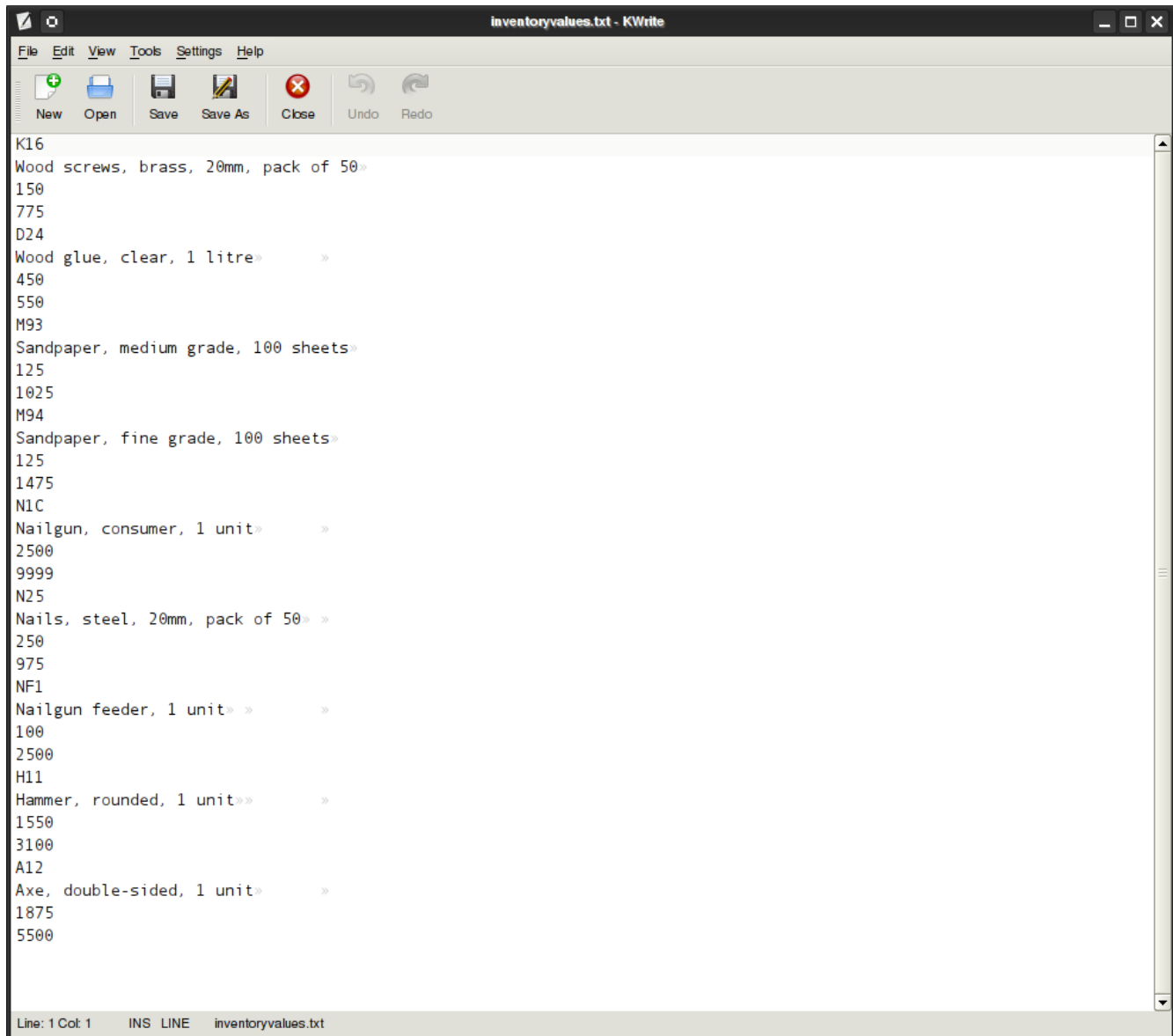
inventoryDisplay.txt: used for displaying the product catalogue when program is run

```
inventoryDisplay.txt - KWrite
File Edit View Tools Settings Help
New Open Save Save As Close Undo Redo
K16»
Wood screws, brass, 20mm, pack of 50
150»
$7.75
D24»
Wood glue, clear, 1 litre
450»
$5.50
M93»
Sandpaper, medium grade, 100 sheets
125»
$10.25
M94»
Sandpaper, fine grade, 100 sheets
125»
$14.75
N1C»
Nailgun, consumer, 1 unit
2500»
$99.99
N25»
Nails, steel, 20mm, pack of 50
250»
$9.75
NF1»
Nailgun feeder, 1 unit
100»
$25.00
H11»
Hammer, rounded, 1 unit
1550»
$31.00
A12»
Axe, double-sided, 1 unit
1875»
$55.00
Line: 27 Col: 15 INS LINE inventoryDisplay.txt
```

Name: Lum Chee Xiang Michael (Lan Zhixiang)
Student ID: 111102750
Assignment 3

Module: Introduction to Java and Object-oriented Programming
Module number: CO1109-01

inventoryvalues.txt: used for obtaining product information from catalogue to be used by program



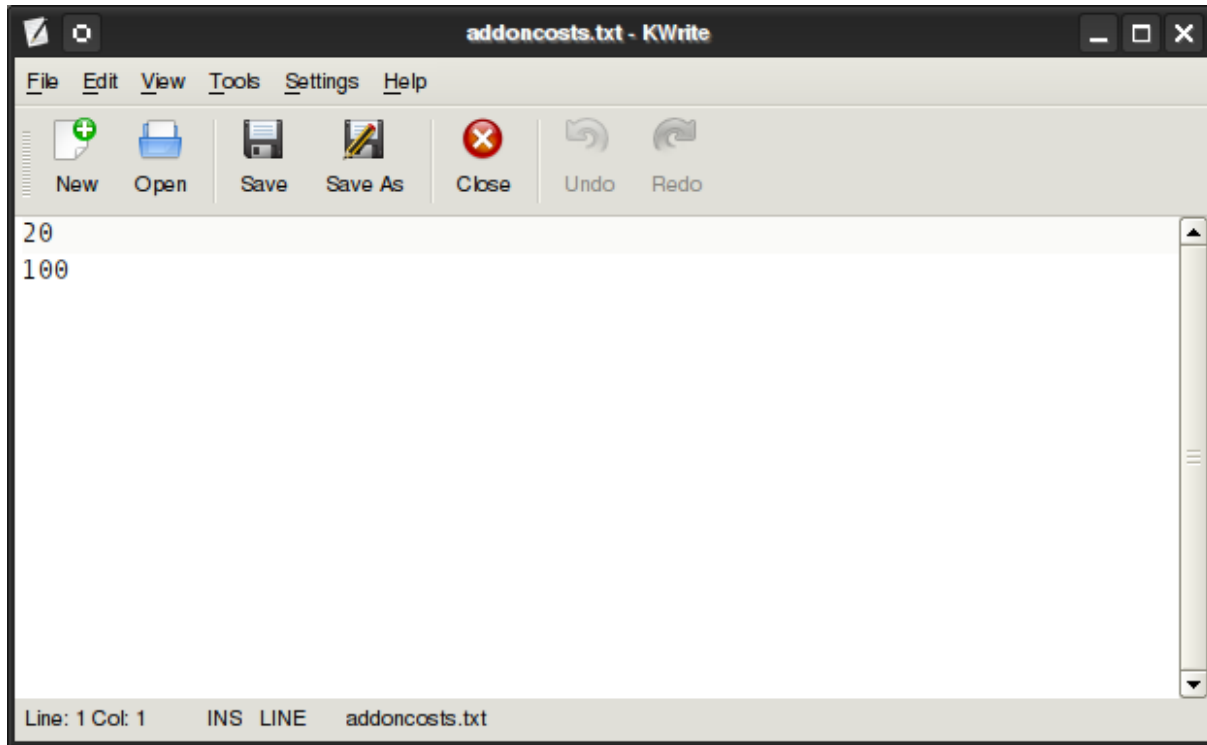
```
K16
Wood screws, brass, 20mm, pack of 50»
150
775
D24
Wood glue, clear, 1 litre»
450
550
M93
Sandpaper, medium grade, 100 sheets»
125
1025
M94
Sandpaper, fine grade, 100 sheets»
125
1475
N1C
Nailgun, consumer, 1 unit»
2500
9999
N25
Nails, steel, 20mm, pack of 50»
250
975
NF1
Nailgun feeder, 1 unit»
100
2500
H11
Hammer, rounded, 1 unit»
1550
3100
A12
Axe, double-sided, 1 unit»
1875
5500
```

*The main difference between the 'inventoryvalues' and 'inventoryDisplay' text files lies in the formatting; 'inventoryDisplay' employs tabs in the product IDs, weight and prices to ensure proper display upon initialization of the program. As such, they cannot be used for storing the values into their respective arrays, for they will cause errors when Integer.parseInt() and Double.parseDouble() is invoked on those values.

Name: Lum Chee Xiang Michael (Lan Zhixiang)
Student ID: 111102750
Assignment 3

Module: Introduction to Java and Object-oriented Programming
Module number: CO1109-01

addoncosts.txt: used for obtaining VAT and shipping costs in calculation of bill



*This file consists of two lines only: the first line contains the VAT information, while the second line displays shipping costs.

Name: Lum Chee Xiang Michael (Lan Zhixiang)
Student ID: 111102750
Assignment 3

Module: Introduction to Java and Object-oriented Programming
Module number: CO1109-01

Customer invoice file:

Invoice_11_03_2012.txt - KWrite

FileEditViewToolsSettingsHelp

New

Open

Save

Save As

Close

Undo

Redo

INVOICE FOR ORDER:

Bill Silic

Apt 23

Building 8

Winchester» POST CODE: 89763

K16»

Wood screws, brass, 20mm, pack of 50»

1 @

\$7.75»

\$7.75

»

150g

D24»

Wood glue, clear, 1 litre»

2 @

\$5.50»

\$11.00

»

900g

»

»

»

»

»

»

Total:

\$18.75

»

»

»

»

»

»

\$5.00

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

»

Name: Lum Chee Xiang Michael (Lan Zhixiang)
Student ID: 111102750
Assignment 3

Module: Introduction to Java and Object-oriented Programming
Module number: CO1109-01

Screen dump of program output

```
Output - Coursework3 (run)
run:
  HARDWARE ITEMS
  CODE      DESCRIPTION      WEIGHT(g)  UNIT PRICE
  K16      Wood screws, brass, 20mm, pack of 50  150      $7.75
  D24      Wood glue, clear, 1 litre            450      $5.50
  M93      Sandpaper, medium grade, 100 sheets  125      $10.25
  M94      Sandpaper, fine grade, 100 sheets   125      $14.75
  N1C      Nailgun, consumer, 1 unit           2500     $99.99
  N25      Nails, steel, 20mm, pack of 50       250      $9.75
  NF1      Nailgun feeder, 1 unit               100      $25.00
  H11      Hammer, rounded, 1 unit             1550     $31.00
  A12      Axe, double-sided, 1 unit            1875     $55.00

Please enter your full name:
> Bill Silic
Please enter your Street address:
> Apt 23
Please enter your House or Apartment number:
> Building 8
Please enter your State of residence:
> Winchester
Please enter your postal code:
> PLACE YOUR ORDER!
NAME: Bill Silic
ADDRESS-1: Apt 23
ADDRESS-2: Building 8
ADDRESS-3: Winchester
POST CODE: 89763

ENTER CODE: (XX to Stop)
CODE:
89763
QUANTITY:
D24
CODE:
2
QUANTITY:
K16
1
CODE:

INVOICE FOR ORDER:
Bill Silic
Apt 23
Building 8
Winchester      POST CODE: 89763

K16      Wood screws, brass, 20mm, pack of 50      1 @ $7.75      $7.75      150g
D24      Wood glue, clear, 1 litre                  2 @ $5.50     $11.00     900g
                                         Total: $18.75
SHIPPING: 1050g @ $1.00 per 250g                                     $5.00
TOTAL INCL. SHIPPING: $23.75
VAT at 20%                                                             $4.75
TOTAL TO PAY: $28.50

A copy of this invoice has been printed.

Is there another customer waiting to perform a transaction on this machine?
xx
Enter N for no, or Y or yes, (followed by the 'Enter key') to proceed.> Y
Please enter your full name:
> Negi Springfield
Please enter your Street address:
> Mahora Academy
```

Name: Lum Chee Xiang Michael (Lan Zhixiang)
Student ID: 111102750
Assignment 3

Module: Introduction to Java and Object-oriented Programming
Module number: CO1109-01

```
Output - Coursework3 (run)

A copy of this invoice has been printed.

Is there another customer waiting to perform a transaction on this machine?
xx
Enter N for no, or Y or yes, (followed by the 'Enter key') to proceed.> Y
Please enter your full name:
> Negi Springfield
Please enter your Street address:
> Mahora Academy
Please enter your House or Apartment number:
> Please enter your State of residence:
Building 3
> Please enter your postal code:
Wales
> 51057
PLACE YOUR ORDER!
NAME: Negi Springfield
ADDRESS-1: Mahora Academy
ADDRESS-2: Building 3
ADDRESS-3: Wales
POST CODE: 51057

ENTER CODE: (XX to Stop)
CODE:
QUANTITY:
N1c
1
CODE:
A12
QUANTITY:
CODE:
1
H11
QUANTITY:
1
CODE:
xx

INVOICE FOR ORDER:
Negi Springfield
Mahora Academy
Building 3
Wales POST CODE: 51057

NIC    Nailgun, consumer, 1 unit          1 @ $99.99    $99.99    2500g
H11    Hammer, rounded, 1 unit          1 @ $31.00    $31.00    1550g
A12    Axe, double-sided, 1 unit         1 @ $55.00    $55.00    1875g
                                           Total: $185.99

SHIPPING: 5925g @ $1.00 per 250g                                $24.00
TOTAL INCL. SHIPPING: $209.99
VAT at 20%                                                        $42.00
TOTAL TO PAY: $251.99

A copy of this invoice has been printed.

Is there another customer waiting to perform a transaction on this machine?
Enter N for no, or Y or yes, (followed by the 'Enter key') to proceed.> Y
Please enter your full name:
> Please enter your Street address:
John Smith
> Smith Ave
Please enter your House or Apartment number:
> Apt #11-14
```

Name: Lum Chee Xiang Michael (Lan Zhixiang)
Student ID: 111102750
Assignment 3

Module: Introduction to Java and Object-oriented Programming
Module number: CO1109-01

```
Output - Coursework3 (run)
TOTAL INCL. SHIPPING: $209.99
VAT at 20%          $42.00
TOTAL TO PAY: $251.99

A copy of this invoice has been printed.

Is there another customer waiting to perform a transaction on this machine?
Enter N for no, or Y or yes, (followed by the 'Enter key') to proceed.> Y
Please enter your full name:
> Please enter your Street address:
John Smith
> Smith Ave
Please enter your House or Apartment number:
> Apt #11-14
Please enter your State of residence:
> London
Please enter your postal code:
> 56485
PLACE YOUR ORDER!
NAME: John Smith
ADDRESS-1: Smith Ave
ADDRESS-2: Apt #11-14
ADDRESS-3: London
POST CODE: 56485

ENTER CODE: (XX to Stop)
CODE:
M93
QUANTITY:
CODE:
1
QUANTITY:
M94
CODE:
1
NF1
QUANTITY:
CODE:
1
CODE:
xx

INVOICE FOR ORDER:
John Smith
Smith Ave
Apt #11-14
London POST CODE: 56485

M93 Sandpaper, medium grade, 100 sheets 1 @ $10.25 $10.25 125g
M94 Sandpaper, fine grade, 100 sheets 1 @ $14.75 $14.75 125g
NF1 Nailgun feeder, 1 unit 1 @ $25.00 $25.00 100g
Total: $50.00
SHIPPING: 350g @ $1.00 per 250g $2.00
TOTAL INCL. SHIPPING: $52.00
VAT at 20% $10.40
TOTAL TO PAY: $62.40

A copy of this invoice has been printed.

Is there another customer waiting to perform a transaction on this machine?
Enter N for no, or Y or yes, (followed by the 'Enter key') to proceed.> N

Thank you for shopping with us. The program will now terminate.
BUILD SUCCESSFUL (total time: 5 minutes 37 seconds)
```

Name: Lum Chee Xiang Michael (Lan Zhixiang)
Student ID: 111102750
Assignment 3

Module: Introduction to Java and Object-oriented Programming
Module number: CO1109-01

Test Description

As stated in the coursework outlines, this program is supposed to be functionally identical to the one that was previously submitted for Coursework 2, with the main exception being that the product catalogue and billing are to be done with data captured from external text files, and that the invoicing should be printed to text files as well.

Therefore, the same functionality tests performed in Coursework 2 will still be relevant, and they will be documented in the form of a table as shown below:

Taking of orders:

Test Scenario	Expected result	Actual result
Nothing ordered; customer returns 'XX' without inputting any values	Program returns 'You have not ordered anything' message, no invoice is displayed or written	Program returns 'You have not ordered anything' message, no invoice is displayed or written
Nothing ordered; customer enters item code but inputs a '0' for quantity (eg: K16 x 0)	Program returns 'You have not ordered anything' message, no invoice is displayed or written	Program returns 'You have not ordered anything' message, no invoice is displayed or written
Extreme value test: customer orders 100 units of every item. Used to test program's integrity in handling extreme values.	Program should be able to calculate the exact bill, as well as display the invoice and save a copy of it to a text file	<div> <div> K16 Wood screws, brass, 20mm, pack of 50 100 @ \$7.75 \$775.00 15000g D24 Wood glue, clear, 1 litre 100 @ \$5.50 \$550.00 45000g W93 Sandpaper, medium grade, 100 sheets 100 @ \$10.25 \$1025.00 125000g W94 Sandpaper, fine grade, 100 sheets 100 @ \$14.75 \$1475.00 125000g N1C Nailgun, consumer, 1 unit 100 @ \$99.99 \$9999.00 250000g N25 Nails, steel, 20mm, pack of 50 100 @ \$9.75 \$975.00 25000g NF1 Nailgun feeder, 1 unit 100 @ \$25.00 \$2500.00 100000g H11 Hammer, rounded, 1 unit 100 @ \$31.00 \$3100.00 155000g A12 Axe, double-sided, 1 unit 100 @ \$55.00 \$5500.00 187500g Total: \$25899.00 </div> <div> SHIPPING: 712500g @ \$1.00 per 250g TOTAL INCL. SHIPPING: \$28749.00 VAT at 20% \$5749.80 TOTAL TO PAY: \$34498.80 </div> </div>
Extreme value test: customer orders 100 units of every item. Same rationale as above.	Program should be able to calculate the exact bill, as well as display the invoice and save a copy of it to a text file	<div> <div> A12 Axe, double-sided, 1 unit 100 @ \$55.00 \$5500.00 187500g Total: \$5500.00 </div> <div> SHIPPING: 187500g @ \$1.00 per 250g TOTAL INCL. SHIPPING: \$6250.00 VAT at 20% \$1250.00 TOTAL TO PAY: \$7500.00 </div> </div> <p>A copy of this invoice has been printed.</p>
Disallowed value test: customer enters an integer that is negative or is greater than 100. Used to test if program is capable of identifying disallowed values	Program prompts user about the disallowed value and asks for re-entry of quantity	ENTER CODE: (XX to Stop) CODE: K16 QUANTITY: 101 Sorry, your total quantity for this product has exceeded 100 or is less than 0. Please re-enter the quantity again:
Standard purchase test: customer orders a reasonable quantity from a selection of products in the catalogue. Used to test if program is able to accept inputs made by the average customer	Program should be able to calculate the exact bill, as well as display the invoice and save a copy of it to a text file	<div> <div> K16 Wood screws, brass, 20mm, pack of 50 1 @ \$7.75 \$7.75 150g N1C Nailgun, consumer, 1 unit 1 @ \$99.99 \$99.99 2500g N25 Nails, steel, 20mm, pack of 50 1 @ \$9.75 \$9.75 250g H11 Hammer, rounded, 1 unit 1 @ \$31.00 \$31.00 1550g A12 Axe, double-sided, 1 unit 1 @ \$55.00 \$55.00 1875g Total: \$203.49 </div> <div> SHIPPING: 6325g @ \$1.00 per 250g TOTAL INCL. SHIPPING: \$229.49 VAT at 20% \$45.90 TOTAL TO PAY: \$275.39 </div> </div>

Name: Lum Chee Xiang Michael (Lan Zhixiang)
Student ID: 111102750
Assignment 3

Module: Introduction to Java and Object-oriented Programming
Module number: CO1109-01

Multiple single order test: customer repeatedly orders differing quantities of a single item (eg: K16 x 2, K16 x 4, K16 x 1, K16 x 5, etc). Used to ensure that program can add the values automatically so that customer does not need to restart the program just to re-enter the correct values	Program should be able to calculate the exact bill, as well as display the invoice and save a copy of it to a text file	<pre> ENTER CODE: (XX to Stop) CODE: K16 QUANTITY: 1 CODE: K16 QUANTITY: 3 CODE: K16 QUANTITY: 2 CODE: INVOICE FOR ORDER: Bill 5111c Apt 28 XX Building 3 Winchester POST CODE: 89763 K16 Wood screws, brass, 20mm, pack of 50 6 @ \$7.75 \$46.50 900g Total: \$46.50 \$4.00 SHIPPING: 900g @ \$1.00 per 250g TOTAL INCL. SHIPPING: \$50.50 VAT at 20% \$10.10 TOTAL TO PAY: \$60.60 A copy of this invoice has been printed. </pre>
Invalid product code test: customer enters an unrecognized product ID	Program should inform customer that the ID is not recognized and keep looping until a suitable ID is keyed	<pre> ENTER CODE: (XX to Stop) CODE: A26 You have entered an invalid code. Please try again. CODE: B99 You have entered an invalid code. Please try again. CODE: </pre>

Program loop test:

Test Scenario	Expected result	Actual result
Another customer performs a transaction after the previous user is done with the program	Program should loop and prompt new customer to enter billing details	<pre> Is there another customer waiting to perform a transaction on this machine? Enter N for no, or Y or yes, (followed by the 'Enter key') to proceed.> Y Please enter your full name: > John Smith Please enter your Street address: > Smith Ave Please enter your House or Apartment number: > #11-14 Please enter your State of residence: > London Please enter your postal code: > PLACE YOUR ORDER! 23689 NAME: John Smith ADDRESS-1: Smith Ave ADDRESS-2: #11-14 ADDRESS-3: London POST CODE: 23689 ENTER CODE: (XX to Stop) CODE: </pre>
No customers are present to perform an additional transaction	Program should terminate	<pre> Is there another customer waiting to perform a transaction on this machine? Enter N for no, or Y or yes, (followed by the 'Enter key') to proceed.> Thank you for shopping with us. The program will now terminate. N BUILD SUCCESSFUL (total time: 15 minutes 37 seconds) </pre>
Customer enters a value that is neither Y, y, yes, N, n, or no	Program should prompt user to enter a proper response and loop	<pre> Is there another customer waiting to perform a transaction on this machine? Enter N for no, or Y or yes, (followed by the 'Enter key') to proceed.> LOL Invalid response. Please try again. > </pre>

Program description

The entire program is broken up into a variety of methods that reside in five classes, mainly the Working class (known as working.java), the Customer class (known as customer.java), the Order class (known as order.java), the Product class (known as product.java) and lastly, the End class (known as end.java).

The Working class is the only class that contains a **public static void main (String[] args)** section; therefore, it serves as the class that will initialize the entire program by making references to various methods written within the other classes. When the working class is executed, it creates an object known as 'o' of the Order class, while employing the FileReader tool to retrieve and display the hardware shop's product catalogue via an external text file known as 'inventoryDisplay.txt'. This catalogue is then combined with the showInventory() method to provide suitable headers for the catalogue's display, as well as using the created object 'o' to invoke three methods from the Object class; readfile(), shipAndtax() and startOrder().

By doing so, the program will make use of FileReader to retrieve information about the various products available in the shop's catalogue from another external text file known as 'inventoryvalues.txt', in which an array of objects known as **stuff** is created by using the constructors defined in the Product class such that various parameters like Product ID and Product Description have their own arrays. Similarly, the shipAndtax() method in the Object class makes use of FileReader to assign values to the VAT and shipping costs variables via the 'addoncosts.txt' file.

This just leaves the startOrder() method, which contains the necessary code needed to facilitate the entire order-taking process that forms the program's main function. This is accomplished by first resetting all purchase values and quantities to 0 as a 'safety' measure of sorts to ensure that no leftover values from a previous transaction is carried forward, while a created object 'a' of the customer class calls the billinfo() and displayCustInfo() methods in the Customer class via the getCustomer() method in the Order class to allow the program to prompt the customer to enter the billing information needed to facilitate a transaction.

Upon the entry of a customer's billing information, the next step in the startOrder() method consists of prompting the customer to enter the ID of the corresponding product he or she wishes to purchase, or to simply enter 'XX' to terminate the order-taking process. A for() loop encapsulated within a while() loop handles this order-taking process: the former is used to simplify the amount of code needed to determine whether the product ID entered by the customer is recognized or not, and if it is, to prompt the customer to enter in a suitable value for the order via the getOrderQuantity() method in the Product class. In the event that a customer keys in a quantity that exceeds 100 units or a negative value, getOrderQuantity() will inform the customer that an invalid value has been entered, reset the order quantity of the corresponding product to 0 and prompt them to re-enter a suitable value. Once a suitable value is added, getOrderQuantity() will make use of the corresponding information stored in the object array to determine the total weight of the order for that specific product while calculating the total cost for it as well.

The while() loop mentioned earlier is used to ensure that the act of prompting the customer to enter a product ID and quantity loops infinitely until a response of 'XX' is returned so that the program does not end up terminating prematurely.

In the event that the customer enters 'XX' to end the order-taking process, the method takeXX() will be called, in which the program will attempt to calculate the total number of products ordered by the customer. If that number equates to 0, the takeNothing() method specified in the if() statement of the method is invoked, in which the program will display a message informing the customer that nothing has been ordered. Otherwise, the program will proceed to the else() section specified in the method, in which two other methods are called, namely writeInvoice() and takeInvoice().

In the case of `writeInvoice()`, a complete copy of the customer's invoice, complete will billing information, shipment charges and total amount payable is generated, just like how it was done previously in Coursework 2. However, the difference lies in the fact that this time, the invoice is not actually displayed by the program, but written to an external text file known as 'Invoice_dd_mm_yyyy.txt', where dd, mm and yyyy refers to the current date of the transaction. This allows the program to generate invoice files based on transaction dates, which in turn allows for easy archival since a new invoice file will be generated for every first transaction that takes place on a new day.

In addition, to prevent a later transaction that occurs in the same day from overwriting the file housing the previous transaction, the boolean 'true' is added to `FileWriter` to ensure that and subsequent transactions that occur in the same day are appended to the bottom of the next file instead of generating their own invoice file that overwrites those of previous transactions. In addition, the `takeInvoice()` method displays the customer's invoice on the program as well so that visual confirmation of the invoice can be done on the customer's end. The `takeInvoice()` also notifies the customer that the transaction has been logged and that the invoice has already been archived by the system.

Lastly, the `startOrder()` method creates an object `n` of the `End` class, upon which the customer is prompted to confirm whether there are other customers waiting to use the system or if there are no more transactions left to be process; this is done via the `endcase()` method in the `End` class. In the event that there are more customers awaiting their turn on the system, `endYes()` is called, and the entire `startOrder()` method loops again to facilitate the processing of the next customer's purchases, otherwise `endNo()` is called and the program subsequently terminates after thanking the customer for shopping. Finally, any response that is not recognized by the program will result in Java looping that specific portion and prompting the customer to enter a valid response.

Problems faced:

Having no experience in any programming language prior to enrolling for this course, this candidate encountered a number of problems in ensuring the successful coding and execution of this program.

The biggest issue that arose was the candidate's tendency to adopt a sequential approach in programming; this can be clearly seen the candidate's previous coursework submissions, in which the entire program was written and coded using procedural programming practices. Because of this, the candidate had a lot of issues coming to terms with the concepts of code modularity, classes, methods and objects. Upon reading the requirements for Coursework 3, the candidate quickly came to the conclusion that procedural programming was no longer an option and was thus forced to adopt a completely different view of programming in order to successfully complete this assignment. This was achieved by reading up on the concepts of object-oriented programming, as well as the aforementioned concepts of modularity, classes, methods and objects. In addition, the candidate also practised converting earlier procedural-written programs into object-oriented programs by attempting to break them up into various methods and classes in order to achieve the level of proficiency required for this coursework.

Another issue faced by the candidate in this Coursework was that of formatting issues across different operating systems where the use of text files was concerned. As this candidate's operating system of choice is a Linux distribution, testing the program across multiple platforms hit a snag when it was discovered that the formatting in a text file created in a Linux-based operating system is lost when the same file is copied to Windows, which in turn causes Java to throw up various exceptions when the program was run in a Windows environment. This issue was solved by maintaining two separate projects which share the same source code, but utilize different external text files created under Linux and Windows respectively.

Yet another issue the candidate had to contend with in the writing of this program was the decision to have `FileReader` read individual characters or entire lines for use in extracting information from the aforementioned external text files. Going with the former would have resulted in the candidate possibly being able to exhibit more control in the invoice's formatting, but would also have introduced additional complexity which the candidate was not confident of tackling due to inexperience. In

Name: Lum Chee Xiang Michael (Lan Zhixiang)
Student ID: 111102750
Assignment 3

Module: Introduction to Java and Object-oriented Programming
Module number: CO1109-01

the end, the decision was made to use Scanner to read the information on a line-by-line basis for simplicity.

Lastly, the candidate also faced various issues in attempting to ensure that the numerical values stored in the external text files which contained information about the product's weight and pricing were properly converted into **int** and **double** values for use in calculating a customer's bill. This was eventually solved by reading up on the features of Integer.parseInt() and Double.parseDouble(), which allowed the candidate to do just that.