```
/**
* I need to be walked through with why the stocks are being recognized "half way."
* They will print out in the console but won't be recognized by certain code.
* Every line of code seems to look right and that's where I'm confused.
*/

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;


/**
* This class builds the GUI window for the Inventory Program and contains the
* main method to get the program started.
*/
public class GUI implements ActionListener {


    private static GUI window; // Link to the GUI object, once created


    /**
     * Call this to get the whole thing started. Creates GUI and Inventory
     * objects.
     *
     * @param args
     * not used.
     */
    public static void main(String[] args) {
        window = new GUI(new Inventory());
    }


    /**
     * Prints the given string to the GUI's text area.
     *
     * @param message
     * the text to print.
     */
    public static void println(String msg) {
        if (window != null) {
            window.display(msg + "\n");
        }
    }


    private Inventory inventory; // Link to the Inventory object
    private JFrame frame; // The window itself
    private JTextArea output; // The window's text area
```

```
private JButton stock; // Button to stock a product
private JButton sell; // Button to sell a product
private JButton print; // Button to print data on all products
private JButton price; // Button to check the price of a product
private JButton total; // Button to compute the total value of products
private JButton purge; // Button to remove all products with quantities of 0
private JMenuItem clearDisplay; // Menu option to clear the text area
private JMenuItem quit; // Menu option to end the program
private JMenuItem add; // Menu option to add a new product
private JMenuItem remove; // Menu option to delete a product


/**
 * Constructor!
 *
 * @param inventory
 * an Inventory object
 */
public GUI(Inventory inventory) {
   this.inventory = inventory;
   inventory.addProduct("Shoes", 20.25);
   inventory.addProduct("Socks", 5.50);
   inventory.addProduct("Pants", 89.99);
   inventory.addProduct("Shirt", 49.00);
   makeFrame();
}


/**
 * Displays the given string in the text area.
 *
 * @param message
 * the text to display.
 */
public void display(String msg) {
   output.append(msg);
}


/**
 * Clears all text from the text area.
 */
public void clear() {
   output.setText("");
}


/**
 * Handles all events (button and menu item clicks).
```

```
 *
 * @param e
 * the event that occurred.
 */
@Override
public void actionPerformed(ActionEvent e) {
   Object event = e.getSource();
   if (event.equals(remove)) { // Remove option clicked
      String name = JOptionPane.showInputDialog(frame,
            "Enter old product name: ");
      if (name != null) {
         double price = getDouble("Enter individual product price: ");
         inventory.removeProduct(name, price);
      }
   } else if (event.equals(total)) { // Compute Total Value button clicked


      inventory.totalValue();


   } else if (event.equals(stock)) { // Stock button clicked


      String name = JOptionPane.showInputDialog(frame,
            "Enter product name: ");
      if (name != null) {
         int number = getInt("How many items are you adding?");
         inventory.stockProduct(name, number);
      }


   } else if (event.equals(sell)) { // Sell button clicked


      String name = JOptionPane.showInputDialog(frame,
            "Enter product name: ");
      if (name != null) {
         int number = getInt("How many items are you selling?");
         inventory.sellProduct(name, number);
      }


   } else if (event.equals(add)) { // Add option clicked


      String name = JOptionPane.showInputDialog(frame,
            "Enter new product name: ");
      if (name != null) {
         double cost = getDouble("Enter individual product price: ");
```

```
            inventory.addProduct(name, cost);
        }


    } else if (event.equals(print)) { // Print button clicked


        inventory.printProducts();


    } else if (event.equals(price)) { // Check Price button clicked


        String name = JOptionPane.showInputDialog(frame,
                "Enter product name: ");
        if (name != null) {
            double cost = inventory.checkPrice(name);
            if (cost == -1) {
                JOptionPane.showMessageDialog(frame, name
                        + " is not a product.");
            } else {
                JOptionPane.showMessageDialog(frame, name + ": $" + cost);
            }
        }


    } else if (event.equals(clearDisplay)) { // Clear option clicked


        clear();
    } else if (event.equals(purge)) { // Purge button clicked
        String answer = JOptionPane
                .showInputDialog(frame,
                        "Would you like to check for products with quantities of 0? (yes or no):
");
        if (answer.equalsIgnoreCase("yes")) {
            inventory.Purge();
            JOptionPane.showMessageDialog(frame,
                    "All products with quantities of 0 removed.");
        } else {
            if (answer.equalsIgnoreCase("no")) {
                JOptionPane
                        .showMessageDialog(frame, "Ok! Have a nice day!");
            } else if (event.equals(quit)) { // Exit option clicked


                JOptionPane.showMessageDialog(frame,
                        "Thank you for using this Valuable program!");
                System.exit(0);
```

```
            } else { // Unknown event occurred


                JOptionPane.showMessageDialog(frame,
                        "Error: " + e.getActionCommand()
                                + " not recognized.");
            }
        }
    }
}


/**
 * Gets an integer from the user. Will keep prompting for an integer until
 * one is entered.
 *
 * @param prompt
 * the prompt to display.
 * @return the integer entered by the user.
 */
private int getInt(String prompt) {
    String input = JOptionPane.showInputDialog(frame, prompt);
    try {
        return Integer.parseInt(input);
    } catch (NumberFormatException e) {
        JOptionPane.showMessageDialog(frame, "Error: " + input
                + " is not an integer.");
        return getInt(prompt);
    }
}


/**
 * Gets a double form the user. Will keep prompting for a double until one
 * is entered.
 *
 * @param prompt
 * the prompt to display.
 * @return the double entered by the user.
 */
private double getDouble(String prompt) {
    String input = JOptionPane.showInputDialog(frame, prompt);
    try {
        return Double.parseDouble(input);
    } catch (NumberFormatException e) {
        JOptionPane.showMessageDialog(frame, "Error: " + input
                + " is not a number.");
```

```
        return getDouble(prompt);
    }
}


/**
 * Builds the GUI window with all buttons and menu items!
 */
private void makeFrame() {
    frame = new JFrame("Keepin' Track o' Stuff");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    Container pane = frame.getContentPane();
    pane.setLayout(new BorderLayout());


    output = new JTextArea();
    pane.add(new JScrollPane(output), BorderLayout.CENTER);


    JMenuBar menuBar = new JMenuBar();
    JMenu file = new JMenu("File");
    add = new JMenuItem("Add new product");
    add.addActionListener(this);
    file.add(add);
    remove = new JMenuItem("Remove product");
    remove.addActionListener(this);
    file.add(remove);
    clearDisplay = new JMenuItem("Clear display");
    clearDisplay.addActionListener(this);
    file.add(clearDisplay);
    quit = new JMenuItem("Exit");
    quit.addActionListener(this);
    file.add(quit);
    menuBar.add(file);
    frame.setJMenuBar(menuBar);


    JPanel left = new JPanel();
    left.setLayout(new GridLayout(5, 1));
    stock = new JButton("Stock");
    stock.addActionListener(this);
    left.add(stock);
    sell = new JButton("Sell");
    sell.addActionListener(this);
    left.add(sell);
    print = new JButton("Print");
    print.addActionListener(this);
    left.add(print);
    price = new JButton("Check Price");
```

```
        price.addActionListener(this);
        left.add(price);
        total = new JButton("Compute Total Value");
        total.addActionListener(this);
        left.add(total);
        pane.add(left, BorderLayout.WEST);
        purge = new JButton("Purge");
        purge.addActionListener(this);
        left.add(purge);


        frame.pack();
        frame.setSize(600, 300);
        frame.setLocationRelativeTo(null);
        frame.setVisible(true);
    }
}
```

```
/**
* An object of this class is a type of product that includes:
*
* 1) The product name 2) The quantity of the product available 3) The
* individual product price
*/
public class Product {


    private String name; // Product name
    private int quantity; // How many items are available
    private double price; // Price of each item


    // CONSTRUCTORS
    /**
     * Provide the name and price.
     *
     * @param name
     * the name of the product.
     * @param price
     * the price of an individual item.
     */
    public Product(String name, double price) {
        this.name = name;
        this.price = price;
        quantity = 0;
    }


    /**
     * Provide name, price and quantity.
     *
     * @param name
     * the name of the product.
     * @param price
     * the price of an individual item.
     * @param quantity
     * the number of products in stock.
     */
    public Product(String name, double price, int quantity) {
        this.name = name;
        this.price = price;
        this.quantity = quantity;
    }


    // ACCESSOR METHODS
    /**
```

```
 * Returns the product's name
 *
 * @return name
 */
public String getName() {
   return name;
}


/**
 * Is there a duplicate of the product being added?
 */
@Override
public boolean equals(Object other) {
   if (other instanceof Product) {
      return true;
   } else {
      if (!(other instanceof Product)) {
         return false;
      }
      Product p = (Product) other;
      if (this.getName().equals(p.getName())
            && this.getPrice() == p.getPrice()) {
         return true;
      } else {
         return false;
      }
   }
}


/**
 * Returns the number of items available.
 *
 * @return quantity
 */
public int getQuantity() {
   return quantity;
}


/**
 * Returns the price of an item.
 *
 * @return price
 */
public double getPrice() {
   return price;
}
```

```
// MUTATOR METHODS
/**
 * Sets a new price for the product.
 *
 * @param newPrice
 * the price.
 */
public void setPrice(double newPrice) {
    price = newPrice;
}


/**
 * Increases the quantity by the given number.
 *
 * @param number
 * number of items added.
 */
public void stock(int number) {
    quantity += number;
}


/**
 * Decreases the quantity by the given number.
 *
 * @param number
 * number of items sold.
 */
public void sell(int number) {
    quantity -= number;
}


/**
 * Clears the inventory.
 */
public void clear() {
    quantity = 0;
}


/**
 * Converts the information into a readable format.
 */
@Override
public String toString() {
```

```
        String s = name + "| $" + price + " | " + quantity;
        return s;
    }
}
```

```
/**
* An object of this class maintains an inventory of Products.
*/
import java.util.ArrayList;


public class Inventory {


    private ArrayList<Product> storage;


    /**
     * Default constructor.
     */
    public Inventory() {
        storage = new ArrayList<Product>();
    }


    /**
     * Stock an existing product. Does nothing if given product is not in
     * inventory.
     *
     * @param name
     * the name of the product.
     * @param moreItems
     * number of items to add.
     */
    public void stockProduct(String name, int moreItems) {
        Product p = findProduct(name);
        if (p != null) {
            p.stock(moreItems);
        }
    }


    /**
     * Sell an existing product. Does nothing if given product is not in
     * inventory.
     *
     * @param name
     * the name of the product.
     * @param itemSold
     * the number of items to sell.
     */
    public void sellProduct(String name, int itemsSold) {
        Product p = findProduct(name);
        if (p != null) {
```

```
        p.sell(itemsSold);
    }
}


/**
 * Prints info on all the products in inventory.
 */
public void printProducts() {
    for (int i = 0; i < storage.size(); i++) {
        System.out.println(storage.get(i));
    }
}


/**
 * Add a new product type to inventory. Does nothing if inventory is full.
 *
 * @param name
 * name of the new product.
 * @param price
 * the price of an individual item.
 */
public void addProduct(String name, double price) {
    int n = storage.indexOf(name);
    if (n < 0) {
        Product p = new Product(name, price);
        storage.add(p);
    }
}


public void removeProduct(String name, double price) {
    int n = storage.indexOf(name);
    if (n >= 0) {
        storage.remove(storage.indexOf(name));
    }
}


/**
 *
 * @return total value of all of the products in the inventory
 */
public double totalValue() {
    double total = 0;
    for (int i = 0; i < storage.size(); i++) {
        Product allItems = (storage.get(i));
        if (allItems != null) {
```

```
            total += allItems.getPrice();
        }
    }
    System.out.println("$" + total);
    return total;
}


/**
 * Returns the price of an existing product.
 *
 * @param name
 * the name of the product.
 * @return the price of the product or -1 if product not in inventory.
 */
public double checkPrice(String name) {
    Product p = findProduct(name);
    if (p != null) {
        return p.getPrice();
    } else {
        return -1;
    }
}


/**
 * Removes all products with a quantity of 0
 */
public void Purge() {
    for (int i = 0; i < storage.size(); i++) {
        if (storage.get(i).getQuantity() == 0) {
            storage.remove(storage.get(i));
        }
    }
}


/**
 * Returns the Product object with the given name.
 *
 * @param name
 * name of the product to find.
 * @return the Product object or null if product not in inventory.
 */
public Product findProduct(String name) {
    for (int i = 0; i < storage.size(); i++) {
        if (storage.contains(name)) {
            return storage.get(i);
        }
```

```
        }
        System.out.println("We are out of stock on that product.");
        return null;
    }
}
```