**Testing Verification Document**

This document confirms and explains the specifications for the program.

|  |  |
| --- | --- |
| **Programmer Names** | Logan Baldwin, Parker Matchett |
| **Completion Date** | 2025-04-11 |

We have tested our project for the following specifications and indicated whether the program works as specified by placing **blue check mark** to indicating meeting the specification or **X** for not working. We have explained how the specification was met or indicated what is missing from the project.

|  |  |  |
| --- | --- | --- |
| **Spec** | **Description** | or **X** |
| **Forms** | Three created, centered on the screen with logo at top right. Both student names on form titles. Same font and design for each. |  |
| **Explain:** | *How data is sent between forms. Type here.*  The cart list ( c ) is sent between frmOrder and frmCheckout by having c be passed through on creation of the form.  (On frmOrder)  frmCheckout cart = new frmCheckout(c);  cart.ShowDialog();  (On frmCheckout)  Cart c;  public frmCheckout(Cart cart)  {  InitializeComponent();  c = cart;  } |  |
| **Login**  **Form** | Class structure of users. Login information read from file. Admin can change products, customers can order. |  |
| **Explain:** | *What happens if: (type here)*   1. *file issues, use of exception handling*   if the user file isn’t found, a file dialog opens prompting the user to search for a txt file to use   1. *user cannot login*   A message box appears asking the user to enter the missing piece of information   1. *admin logs in*   frmOrder opens with tabOrder set to “Admin”, revealing buttons used for editing the inventory file   1. *customer logs in*   frmOrder opens with tabOrder set to “Customer”, revealing buttons used add items to or view the cart |  |
| **Static Classes** | Static classes used for files and data validation. |  |
| **Explain:** | The static class FileHelper has the methods ReadUser and ReadProducts which read through files and turn them into lists of users and products.  (Example with ReadProducts)  for (int i = 0; i < size; i++)  {  arr = reader.ReadLine().Split(',');  if (Validator.ValidateItem(arr, out code, out price, out qty, out age))  {  if (arr[0] == "furniture")  {  inventorycart.Add(new AntiqueFurniture(code, arr[2], price, qty, arr[5], arr[6]));  }  else if (arr[0] == "jewelry")  {  inventorycart.Add(new VintageJewelry(code, arr[2], price, qty, age, arr[6]));  }  }  }  reader.Close(); |  |
| **Product** | Class structure of products. Reading from file. |  |
| **Explain:** | *What happens if: Type here.*   1. *file issues, use of exception handling*   if the inventory file isn’t found, a file dialog opens prompting the user to search for a txt file to use   1. *how products are displayed to choose*   The products description and quantity are displayed in a list box. When one is selected, other information like its price, age, or origin are displayed in a label to the right |  |
| **Operator**  **Overloading** | Overload a mathematical and one relational operator |  |
| **Explain:** | *Math operator overloaded and when it is called. Type here.*  The + operator has been overloaded for the cart and is used to create a new cart with the specified product inside it  *Relational operator overloaded and when it is called. Type here.*  The < and > operators have been overloaded for products and are used to clone objects |  |
| **Structures** | One list, using polymorphism to manage the collection. |  |
| **Explain:** | *List: type, scope, use and how it is updated and displayed. Type here.*  Stock is stored in the list inventory from the public class inventory. It gets updated by FileHelper when frmOrder opens, when an admin adds/deletes/edits an item, or when a customer buys an item.  Its displayed in the frmOrder listbox by using inventory’s UpdateListBox method |  |
| **Interface** | One built-in interface OR created interface. |  |
| **Explain:** | *Interface chosen, what it does and how it is called within program. Type here.*  We use the interface IUpdater on the Inventory and Cart classes, which gives them the Clear, UpdateListBox, and HideZeroes methods that they use to interact with listboxes in the forms. |  |
| **Checkout**  **Form** | Customer has selected items and is ready to place order. |  |
| **Explain:** | *How check out form works, class and file populated, how products get updated. Type here.*  The cart list made in frmOrder gets passed to frmCheckout and the listbox is populated by calling the cart class’s UpdateListBox method. When the user purchases the items, FileHelper is used to edit the inventory file to update the stock of the item they bought. |  |
| **Design** | Design consistent throughout all forms (font, layout). |  |
| **Explain:** | The design of the program was kept fairly simple. All forms use Microsoft sans serif as the font, buttons are given bold text, each form has the logo in the top right corner. For layout, input is to the left with additional information in labels to the right and buttons on the bottom. |  |
| **Repeated Code** | Created methods for repeated code. |  |
| **Explain:** | *List methods. Type here*  frmOrder has the method FillLabels() which updates the labels for cart cost and items in cart whenever the cart is changed. frmCheckout has the methods FillLabels() and ClearAll(). FillLabels() calculates the subtotal, tax, and total cost of the cart and puts that information into the labels on the right. ClearAll() clears the listbox and labels when items are paid for or the user clears their cart. If the user cleared their cart instead of paying, it returns the items back into the inventory. |  |
| **Unit Testing** | Test one method with two pathways in one of the classes. |  |
| **Explain:** | *List class and method selected. Type here.*  TotalCartTest tests the TotalCart() method from the class cart |  |
| **Submission** | Includes 4 files: memo, zip of solution, video of test cases, testing verification document |  |