Java Scope Assignment: Understanding Variable Accessibility

In this assignment you will practice your understanding of scope, read and follow code, and debug scope errors..

Part 1: Output Prediction & Scope Analysis (Approx. 40 minutes)

Objective: Analyze the given Java code snippets, predict their exact output, and thoroughly identify the accessibility of variables, including instances of shadowing.
Instructions:
For each code snippet below:

1. Predict what will be printed to the console, line by line.
2. For each variable declared in the snippet, state its scope (where in the code it is accessible) and note any instances of variable shadowing.

Snippet A: Nested Conditionals & Loops

```
public class ScopePredictorA {
    public static void main(String[] args) {
        int globalCounter = 0;
        String status = "Initial";

        while (globalCounter < 4) {
            int loopValue = globalCounter * 10;
            System.out.println("Outer Loop: globalCounter=" + globalCounter + ", loopValue=" +
loopValue);

            if (globalCounter % 2 == 0) {
                String statusMessage = "Even iteration";
                System.out.println("  Inner If: " + statusMessage);
                int tempValue = globalCounter + 5;
                System.out.println("  Inner If Temp: " + tempValue);
            } else {
                String statusMessage = "Odd iteration"; // This 'statusMessage' shadows the one in
the 'if' block
                System.out.println("  Inner Else: " + statusMessage);
                // Can tempValue be accessed here?
                // System.out.println(tempValue); // Would cause an error
            }
```

```
            // Can statusMessage be accessed here?
            // System.out.println(statusMessage); // Would cause an error

            globalCounter++;
        }
        System.out.println("End of Snippet A. Final globalCounter: " + globalCounter + ", status: " +
status);
    }
}
```

Predicted Output for Snippet A:
<!---->

Variable Accessibility for Snippet A:

- globalCounter Accessible anywhere inside the main method.
- status Accessible anywhere inside the main method.
- loopValue Accessible anywhere inside the while loop (only for that loop run).
- statusMessage (inside if) Accessible only inside that if block.
- statusMessage (inside else) Accessible only inside that else block.
- tempValue → Accessible only inside that if block where it's made

Snippet B: Shadowing and Re-declaration

```
public class ScopePredictorB {
    public static void main(String[] args) {
        int x = 10;
        System.out.println("Main method x: " + x);

        if (x > 5) {
            int x = 20; // This 'x' shadows the outer 'x'
            System.out.println("Inside if block x: " + x);
            String message = "Condition met";
            System.out.println("Message: " + message);
        }

        // Can the 'x' from the if block be accessed here?
        // System.out.println("After if block x: " + x); // This would refer to the outer x
        // Can 'message' be accessed here?
        // System.out.println(message); // Would cause an error

        int count = 0;
        while (count < 2) {
            String message = "Loop iteration " + count; // This 'message' shadows the one in the 'if'
block
            System.out.println("Inside while loop message: " + message);
```

```
            count++;
        }

        System.out.println("End of Snippet B. Final main method x: " + x);
    }
}
```

Predicted Output for Snippet B:
<!-- Your prediction here -->

Variable Accessibility for Snippet B:

- x (main method) → Anywhere inside the main method.
- x (inside if) → Only inside that if block
- message (inside if) → Only inside that if block.
- count → Anywhere inside the main method
- message (inside while) → Only inside that while

Part 2: Debugging - Advanced Scope Challenges (Approx. 2 hours)

Objective: Identify and correct multiple, often subtle, scope-related errors in a more complex Java program to make it function as intended.
Problem Description:
You are given a Java program designed to manage a simple "Inventory System." It should:

1. Initialize a totalInventoryValue to 0.0.
2. Allow a user to add a specified number of items.
3. For each item, it should:
    - Prompt for the item's name (String).
    - Prompt for the item's price (double).
    - Prompt for the item's quantity (int).
    - Calculate the itemTotal for the current item (price * quantity).
    - Add itemTotal to the totalInventoryValue.
    - If the itemTotal for a single item exceeds a certain threshold (e.g., $100.0), it should print a special message indicating a "high-value item."
    - Keep a separate count of highValueItemCount.
4. After all items are added, it should print the final totalInventoryValue and highValueItemCount.
5. It should also print a summary of the *last* item added (its name, price, and quantity).

The current code has numerous scope issues that prevent it from compiling or running correctly, or from producing the correct output. Your task is to fix all these issues.
Original Code (with errors):

```
import java.util.Scanner; // Needed for user input
```

```
public class InventoryManagerDebugger {
    public static void main(String[] args) {
        double totalInventoryValue = 0.0;
        int maxItems = 3;
        int itemsProcessed = 0;

        Scanner scanner = new Scanner(System.in);

        while (itemsProcessed < maxItems) {
            String itemName; // This should be accessible later
            double itemPrice; // This should be accessible later
            int itemQuantity; // This should be accessible later

            System.out.println("\nEnter details for Item #" + (itemsProcessed + 1) + ":");
            System.out.print("Enter item name: ");
            itemName = scanner.nextLine();

            System.out.print("Enter item price: ");
            itemPrice = scanner.nextDouble();

            System.out.print("Enter item quantity: ");
            itemQuantity = scanner.nextInt();
            scanner.nextLine(); // Consume newline left-over

            double itemTotal = itemPrice * itemQuantity;
            totalInventoryValue += itemTotal;

            double highValueThreshold = 100.0;
            int highValueItemCount = 0; // This is being re-initialized!

            if (itemTotal > highValueThreshold) {
                highValueItemCount++;
                System.out.println("  *** High-value item detected! ***");
            }

            itemsProcessed++;
        }

        // These variables are out of scope here!
        // System.out.println("\n--- Inventory Summary ---");
        // System.out.println("Total Inventory Value: $" + totalInventoryValue);
        // System.out.println("Number of High-Value Items: " + highValueItemCount);
        // System.out.println("Last Item Added: " + itemName + " (Price: $" + itemPrice + ",
Quantity: " + itemQuantity + ")");
```

```
        scanner.close();
        System.out.println("\nInventory processing complete.");
    }
}
```

Instructions:

1. Copy the Original Code into your IDE.
2. Identify all scope-related errors. Consider:
    - Variables being re-initialized inside the loop.
    - Variables declared inside a block that need to be accessed outside that block.
    - Variables that need to retain their value across loop iterations.
    - Variables that need to be accessible *after* the loop.
3. Modify the code to fix these issues so that:
    - totalInventoryValue correctly accumulates the value of all items.
    - highValueItemCount correctly counts the number of high-value items across all iterations.
    - The itemName, itemPrice, and itemQuantity of the *last* item added are correctly stored and accessible after the loop.
    - All summary print statements after the loop compile and display the correct final values.
4. Add clear, detailed comments to your corrected code explaining each fix you implemented and why it was necessary from a scope perspective.

Corrected Code:

```
 import java.util.Scanner;

public class InventoryManagerDebugger {
    public static void main(String[] args) {
        double totalInventoryValue = 0.0;
        int maxItems = 3;
        int itemsProcessed = 0;

        int highValueItemCount = 0; // Count high-value items here

        String itemName = "";       // Save last item name
        double itemPrice = 0.0;     // Save last item price
        int itemQuantity = 0;       // Save last item quantity

        Scanner scanner = new Scanner(System.in);

        while (itemsProcessed < maxItems) {
            System.out.println("\nEnter details for Item #" + (itemsProcessed + 1) + ":");

            System.out.print("Enter item name: ");
```

```
            itemName = scanner.nextLine();

            System.out.print("Enter item price: ");
            itemPrice = scanner.nextDouble();

            System.out.print("Enter item quantity: ");
            itemQuantity = scanner.nextInt();
            scanner.nextLine(); // Clear enter key

            double itemTotal = itemPrice * itemQuantity;
            totalInventoryValue += itemTotal;

            double highValueThreshold = 100.0;

            if (itemTotal > highValueThreshold) {
                highValueItemCount++;
                System.out.println("  *** High-value item! ***");
            }

            itemsProcessed++;
        }

        // Print results after all items
        System.out.println("\n--- Inventory Summary ---");
        System.out.println("Total Value: $" + totalInventoryValue);
        System.out.println("High-Value Items: " + highValueItemCount);
        System.out.println("Last Item: " + itemName + " (Price: $" + itemPrice + ", Quantity: " +
itemQuantity + ")");

        scanner.close();
        System.out.println("\nDone.");
    }
}
```

Submission Guidelines:

- For Part 1, submit a text file or document containing your predicted outputs and detailed variable accessibility explanations for Snippets A and B.
- For Part 2, submit your InventoryManagerDebugger.java file with the corrected code and thorough, well-commented explanations for each fix.

Good luck with this challenging assignment!