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:

This Java program loops 4 times, with globalCounter starting at 0 and incrementing until it reaches 4. In each iteration, loopValue is printed as globalCounter multiplied by 10. When globalCounter is even, the program prints "Even iteration" and calculates tempValue, but tempValue is scoped only within the if block and isn't accessible later. For odd iterations, it prints "Odd iteration" but cannot access tempValue from the even block. statusMessage is also scoped within the if and else blocks, so it is inaccessible outside of them, leading to no changes in status, which remains "Initial" throughout the loop. By the end of the program, globalCounter reaches 4, but status stays unchanged. The final output will show the loop's progress with the calculated loopValue for each iteration, and the message: "End of Snippet A. Final globalCounter: 4, status: Initial."

Variable Accessibility for Snippet A:

- globalCounter: Accessible anywhere inside the main function.
- status: <!-- Where is it accessible? -->
- loopValue: Accessible anywhere inside the While loop
- statusMessage (inside if): <!-- Where is it accessible? -->
- statusMessage (inside else): <!-- Where is it accessible? -->
- tempValue: <!-- Where is it accessible? -->

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:

In Snippet B, the program begins by declaring an x in the main method with an initial value of 10, which is printed. When the program enters the if block, it declares a new x with a value of 20, shadowing the outer x within that scope. This inner x is only accessible inside the if block, so when the program prints it, it shows 20. The message "Condition met" is also printed inside the if block, but this message is scoped only within the block and cannot be accessed after the block ends. Outside the if block, the original x (which remains 10) is used, though attempts to access the message from the if block would result in an error, as it no longer exists. The while loop then declares its own message, which shadows the one in the if block. This message is scoped within the loop and is printed twice, once for each iteration of the loop. By the end of the program, the final x printed is still the original 10, unaffected by the inner x in the if block. The program's behavior demonstrates Java's scoping rules, where variables are confined to the block in which they are declared, leading to variable shadowing and isolation across different parts of the code.

Variable Accessibility for Snippet B:

- x (main method): <!-- Where is it accessible? -->
- x (inside if): <!-- Where is it accessible? -->
- message (inside if): <!-- Where is it accessible? -->
- count: <!-- Where is it accessible? -->
- message (inside while): <!-- Where is it accessible? →

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; // 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);

        int highValueItemCount = 0; // Moved this outside the loop to persist across iterations

        String lastItemName = ""; // To store the last item's name
        double lastItemPrice = 0.0; // To store the last item's price
        int lastItemQuantity = 0; // To store the last item's quantity

        while (itemsProcessed < maxItems) {
            String itemName; // This is still declared inside the loop
            double itemPrice; // This is still declared inside the loop
            int itemQuantity; // This is still declared inside the loop

            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;

            // Check if it's a high-value item
            double highValueThreshold = 100.0;
```

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

            // Store details of the last item
            lastItemName = itemName;
            lastItemPrice = itemPrice;
            lastItemQuantity = itemQuantity;

            itemsProcessed++;
        }

        // After the loop ends, print the inventory summary
        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: " + lastItemName + " (Price: $" + lastItemPrice + ", Quantity:
" + lastItemQuantity + ")");

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

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!