Name : Naisargi Patel | ID : 202201291

Code 1

Q:1 How many errors are there in the program? Mention the errors you have identified
There are two types of error :
1. Data reference error
2. Computational error

Q : 2 Which category of program inspection would you find more effective?

For this code, Computational error were most effective.

Q : 3 Which type of error you are not able to identified using the program inspection?

By using program inspection method

Q : 4 Is the program inspection technique is worth applicable?

No, as the program length is small.

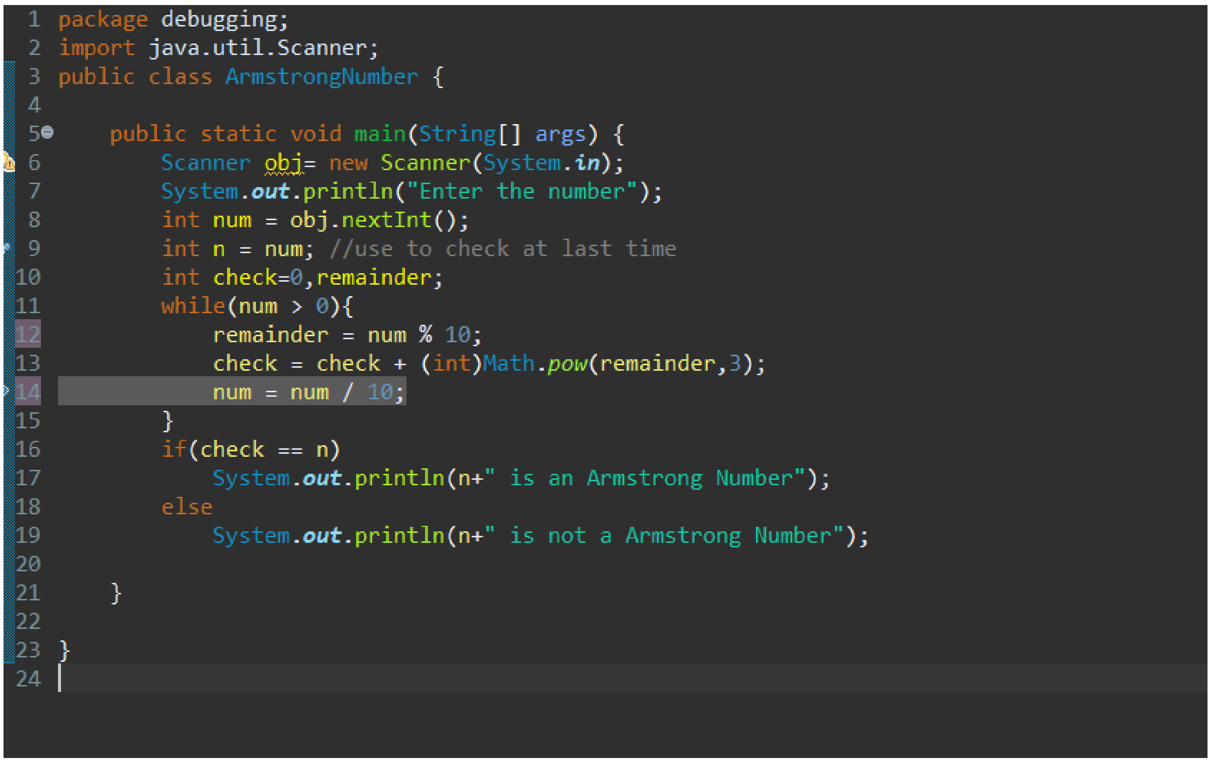
Q : 1 How many errors are there in the program? Mention the errors you have identified.

The code has logic errors in calculating remainder and updating num, plus a grammatical mistake in the output message.

 Q : 2 How many breakpoints you need to fix those errors? What are the steps you have taken to fix the error you identified in the code fragment?

Set breakpoints after the input and within the while loop; correct the logic for calculating remainder and updating num, and fix the output message.

Q : 3



```java
package debugging;
import java.util.Scanner;
public class ArmstrongNumber {

    public static void main(String[] args) {
        Scanner obj= new Scanner(System.in);
        System.out.println("Enter the number");
        int num = obj.nextInt();
        int n = num; //use to check at last time
        int check=0,remainder;
        while(num > 0){
            remainder = num % 10;
            check = check + (int)Math.pow(remainder,3);
            num = num / 10;
        }
        if(check == n)
            System.out.println(n+" is an Armstrong Number");
        else
            System.out.println(n+" is not a Armstrong Number");

    }

}

```

Code : 2

How many errors are there in the program? Mention the errors you have identified.

The program has two main errors. First, in the gcd method, the condition in the while loop is incorrect; it should be while (a % b != 0) instead of while (a % b == 0). Second, the lcm method is not calculating the least common multiple correctly; it currently checks if a is not a multiple of x and y, which should instead check if a is a multiple of both.

Which category of program inspection would you find more effective?

Code reviews tend to be the most effective form of program inspection, as they allow multiple developers to examine the code together, discuss potential issues, and share insights on best practices.

Which type of error you are not able to identified using the program inspection?

Program inspections often miss logical errors or edge cases, such as how the code behaves with negative inputs or zero values, which could cause incorrect outputs or infinite loops.

Is the program inspection technique is worth applicable?

Yes, program inspection techniques are worthwhile as they help identify defects early in the development process, improve code quality, and enhance team collaboration. However, they should be complemented with other techniques like testing and debugging for comprehensive quality assurance.

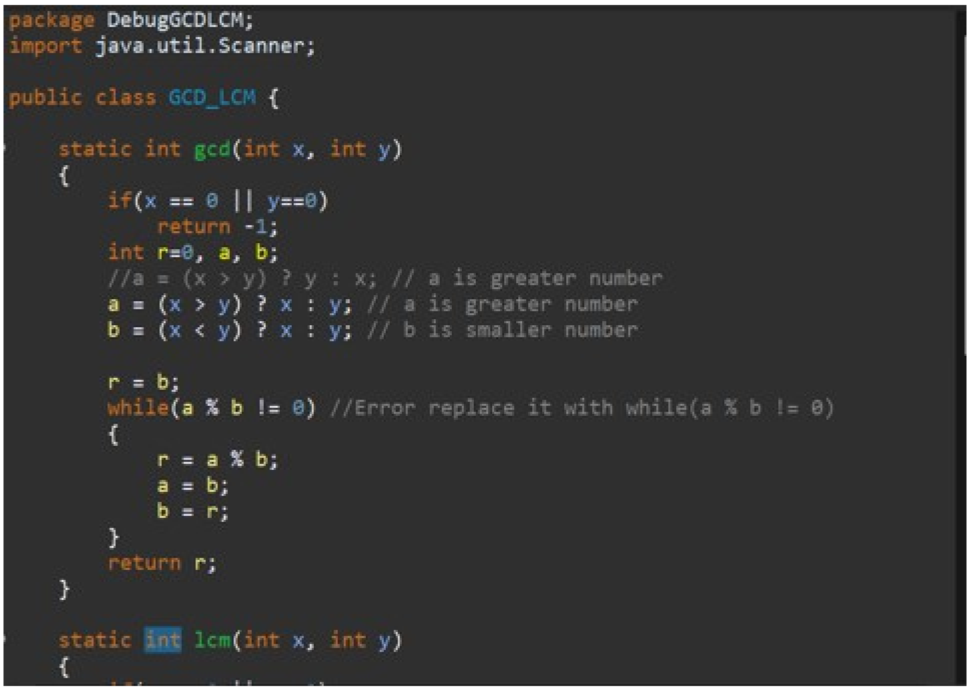
How many errors are there in the program? Mention the errors you have identified.

There are two main errors. First, in the gcd method, the loop condition should be while (a % b != 0) instead of while (a % b == 0). Second, in the lcm method, the condition if (a % x != 0 && a % y != 0) should be corrected to check for multiples; it should be if (a % x == 0 && a % y == 0).

How many breakpoints you need to fix those errors? What are the steps you have taken to fix the error you identified in the code fragment?

set at the beginning of the gcd and lcm methods to inspect variable values and flow of execution. To fix the errors, change the while condition in the gcd method and adjust the condition in the lcm method to correctly check for multiples.

Submit your complete executable code?



```java
package DebugGCDLCM;
import java.util.Scanner;

public class GCD_LCM {

    static int gcd(int x, int y)
    {
        if(x == 0 || y==0)
            return -1;
        int r=0, a, b;
        //a = (x > y) ? y : x; // a is greater number
        a = (x > y) ? x : y; // a is greater number
        b = (x < y) ? x : y; // b is smaller number

        r = b;
        while(a % b != 0) //Error replace it with while(a % b != 0)
        {
            r = a % b;
            a = b;
            b = r;
        }
        return r;
    }

    static int lcm(int x, int y)
    {
```

Code 3

1. How many errors are there in the program? Mention the errors you have identified.

There are multiple errors in the program. First, in the nested loop for filling the opt array, the line int option1 = opt[n++][w]; incorrectly increments n, causing incorrect indexing. Second, in option2, the index should use profit[n] instead of profit[n-2], and there’s a potential ArrayIndexOutOfBoundsException when accessing weight[n] if n exceeds the loop limits.

2. Which category of program inspection would you find more effective?

identifying logical errors

3. Which type of error you are not able to identified using the program inspection?

Program inspections might not identify runtime errors, such as ArrayIndexOutOfBoundsException or logical errors in algorithm implementation that depend on correct indexing.

4. Is the program inspection technique is worth applicable?

Yes, program inspection techniques are worthwhile, as they help catch issues early, improve code quality, and foster collaboration. However, they should be complemented by testing and debugging to ensure comprehensive error detection.

1. How many errors are there in the program? Mention the errors you have identified.

In the line int option1 = opt[n++][w];, the increment of n is incorrect and causes index issues.

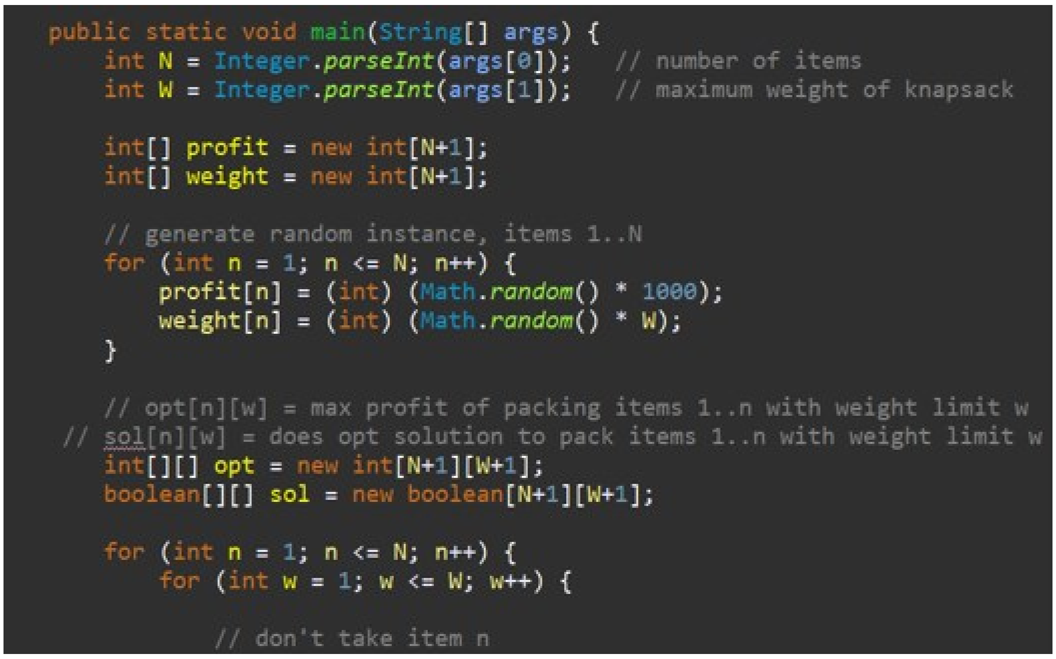
In the calculation of option2, the correct index should be profit[n] instead of profit[n-2].

There’s a potential ArrayIndexOutOfBoundsException when accessing weight[n].

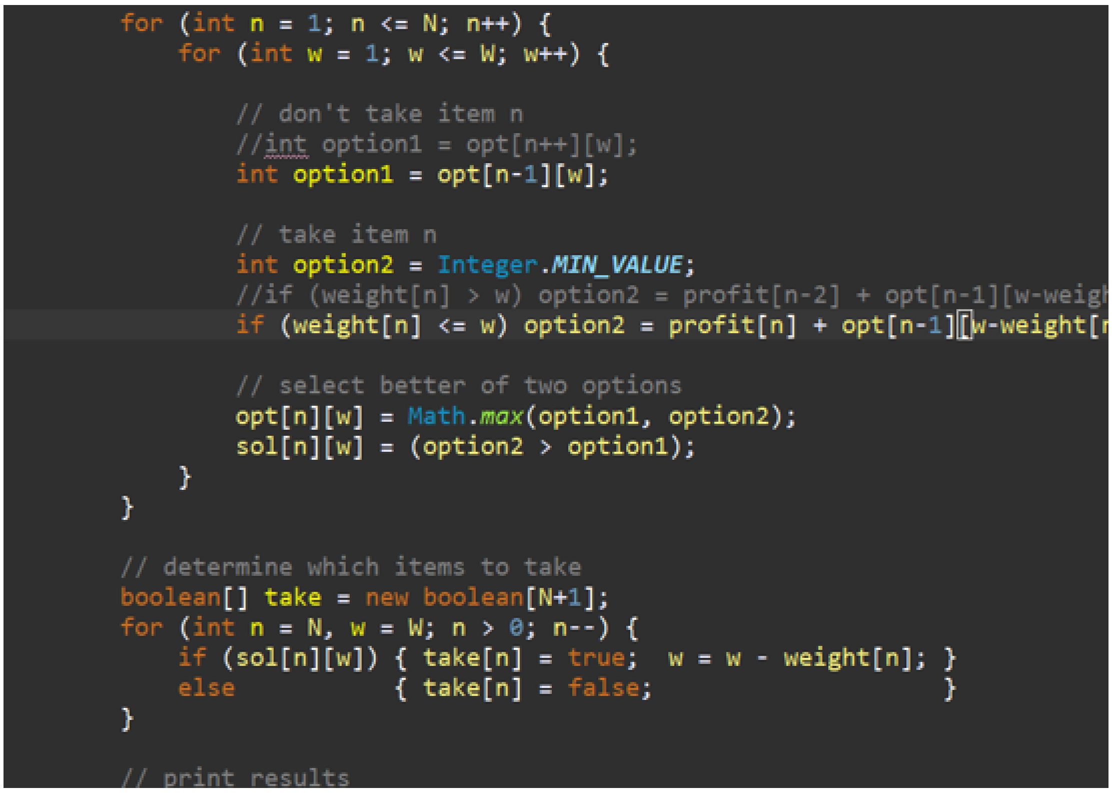
2. How many breakpoints you need to fix those errors? What are the steps you have taken to fix the error you identified in the code fragment?

You need breakpoints at the beginning of the nested loop and where you calculate option1 and option2. To fix the errors, change int option1 = opt[n++][w]; to int option1 = opt[n][w]; and replace profit[n-2] with profit[n].

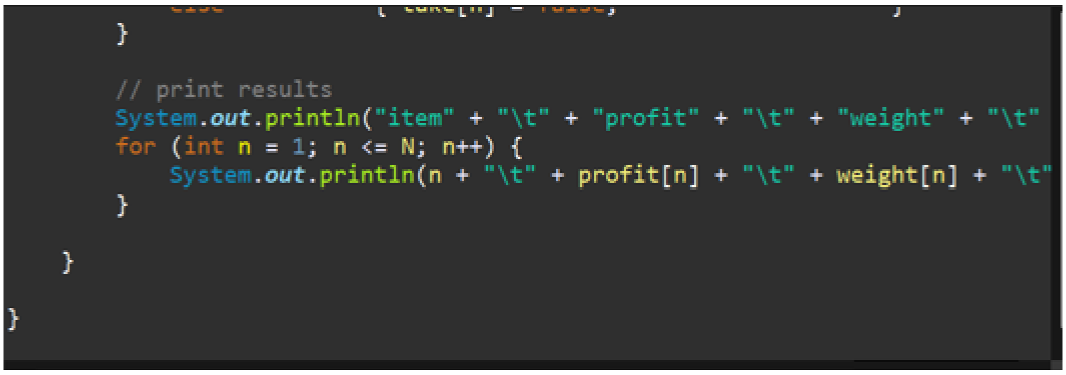
3. Submit your complete executable code?



```
  public static void main(String[] args) {
      int N = Integer.parseInt(args[0]);   // number of items
      int W = Integer.parseInt(args[1]);   // maximum weight of knapsack

      int[] profit = new int[N+1];
      int[] weight = new int[N+1];

      // generate random instance, items 1..N
      for (int n = 1; n <= N; n++) {
          profit[n] = (int) (Math.random() * 1000);
          weight[n] = (int) (Math.random() * W);
      }

      // opt[n][w] = max profit of packing items 1..n with weight limit w
    // sol[n][w] = does opt solution to pack items 1..n with weight limit w
      int[][] opt = new int[N+1][W+1];
      boolean[][] sol = new boolean[N+1][W+1];

      for (int n = 1; n <= N; n++) {
          for (int w = 1; w <= W; w++) {

              // don't take item n
```



```
        for (int n = 1; n <= N; n++) {
            for (int w = 1; w <= W; w++) {

                // don't take item n
                //int option1 = opt[n++][w];
                int option1 = opt[n-1][w];

                // take item n
                int option2 = Integer.MIN_VALUE;
                //if (weight[n] > w) option2 = profit[n-2] + opt[n-1][w-weig
                if (weight[n] <= w) option2 = profit[n] + opt[n-1][w-weight[r

                // select better of two options
                opt[n][w] = Math.max(option1, option2);
                sol[n][w] = (option2 > option1);
            }
        }

        // determine which items to take
        boolean[] take = new boolean[N+1];
        for (int n = N, w = W; n > 0; n--) {
            if (sol[n][w]) { take[n] = true;  w = w - weight[n]; }
            else           { take[n] = false;                    }
        }

        // print results
```



```
        }

        // print results
        System.out.println("item" + "\t" + "profit" + "\t" + "weight" + "\t"
        for (int n = 1; n <= N; n++) {
            System.out.println(n + "\t" + profit[n] + "\t" + weight[n] + "\t"
        }

    }

}
```

Code 5

1. How many errors are there in the program? Mention the errors you have identified.

The program has several errors: the inner while loop condition should be while (sum > 0), not while (sum == 0). Inside this loop, the multiplication should be s += sum % 10; instead of s = s * (sum / 10);, and the statement sum = sum % 10; is missing a semicolon. These errors lead to incorrect logic and potential infinite loops.

2. Which category of program inspection would you find more effective?

Code reviews would be the most effective category of program inspection, as they allow multiple developers to collaboratively identify logical errors and ensure adherence to best practices.

3. Which type of error you are not able to identified using the program inspection?

Program inspections may not identify runtime errors, such as logical flaws in the algorithm or infinite loops, especially if the input cases aren't thoroughly tested.

4. Is the program inspection technique is worth applicable?

Yes, program inspection techniques are worthwhile because they can catch many types of errors early, improve code quality, and facilitate knowledge sharing among team members. However, they should be complemented by testing and debugging for comprehensive error detection.

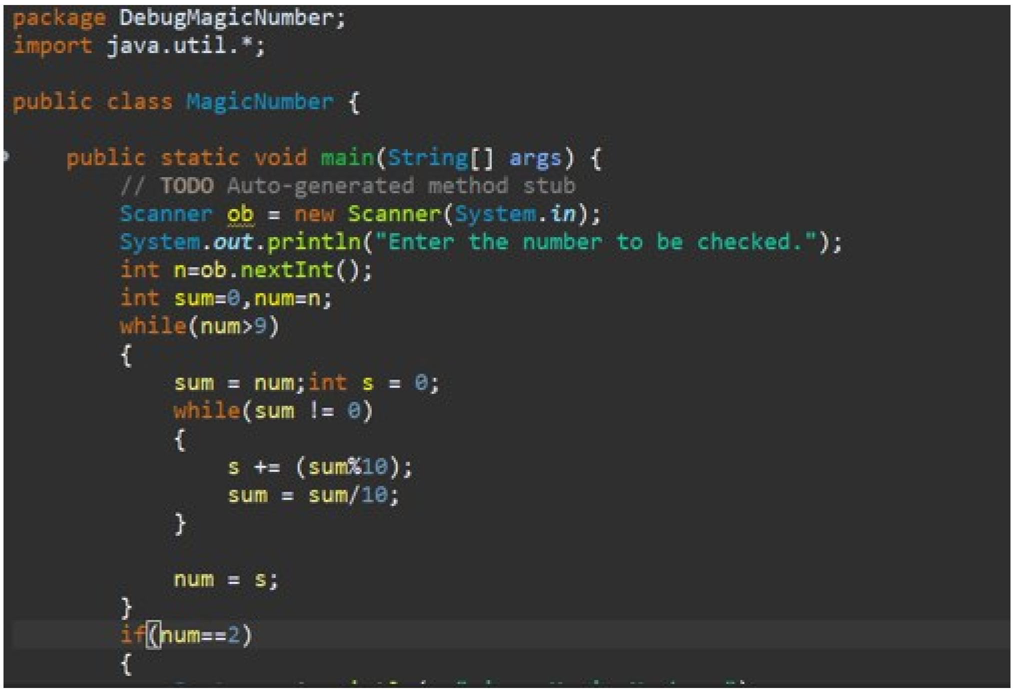
1. How many errors are there in the program? Mention the errors you have identified.

The program contains three main errors. First, the condition in the inner while loop should be while (sum > 0) instead of while (sum == 0). Second, the multiplication statement should be s += sum % 10; instead of s = s * (sum / 10);, which is incorrect for summing the digits. Lastly, the statement sum = sum % 10; is missing a semicolon, leading to a syntax error.

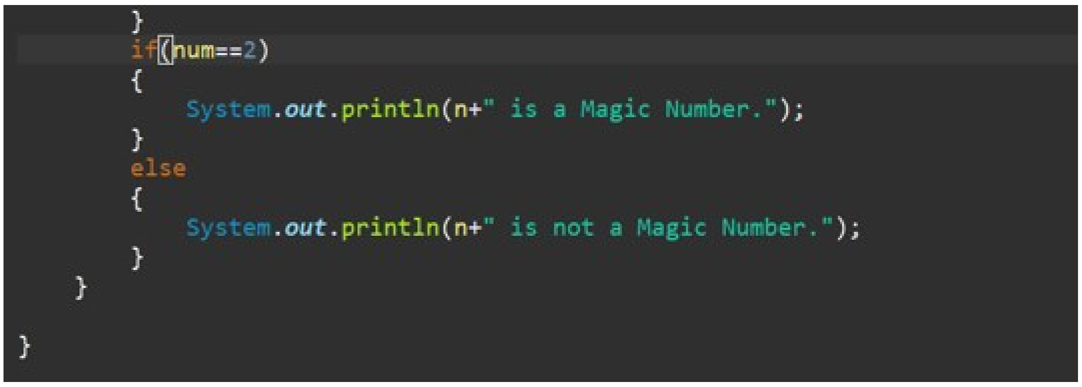
2. How many breakpoints you need to fix those errors? What are the steps you have taken to fix the error you identified in the code fragment?

need breakpoints at the start of the while(num > 9) loop and within the inner loop to inspect the values of sum and s. To fix the errors, change the inner loop condition to while (sum > 0), update the summation logic to s += sum % 10;, and add the missing semicolon after sum = sum / 10;

3. Submit your complete executable code?



```
package DebugMagicNumber;
import java.util.*;

public class MagicNumber {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Scanner ob = new Scanner(System.in);
        System.out.println("Enter the number to be checked.");
        int n=ob.nextInt();
        int sum=0,num=n;
        while(num>9)
        {
            sum = num;int s = 0;
            while(sum != 0)
            {
                s += (sum%10);
                sum = sum/10;
            }

            num = s;
        }
        if(num==2)
        {
```



```
        }
        if(num==2)
        {
            System.out.println(n+" is a Magic Number.");
        }
        else
        {
            System.out.println(n+" is not a Magic Number.");
        }
    }

}
```

Code 6

1. How many errors are there in the program? Mention the errors you have identified.

The program has multiple errors, including incorrect calls to leftHalf and rightHalf using array + 1 and array - 1, invalid usage of left++ and right-- in the merge method, and the uninitialized result array in merge.

2. Which category of program inspection would you find more effective?

Code reviews are the most effective, allowing for collaborative error identification.

3. Which type of error you are not able to identified using the program inspection?

Program inspections may miss performance issues and edge cases, such as handling empty or very large arrays.

4. Is the program inspection technique is worth applicable?

Yes, program inspection techniques are valuable for early error detection and improving code quality but should be combined with testing and debugging for thorough quality assurance.

1. How many errors are there in the program? Mention the errors you have identified.

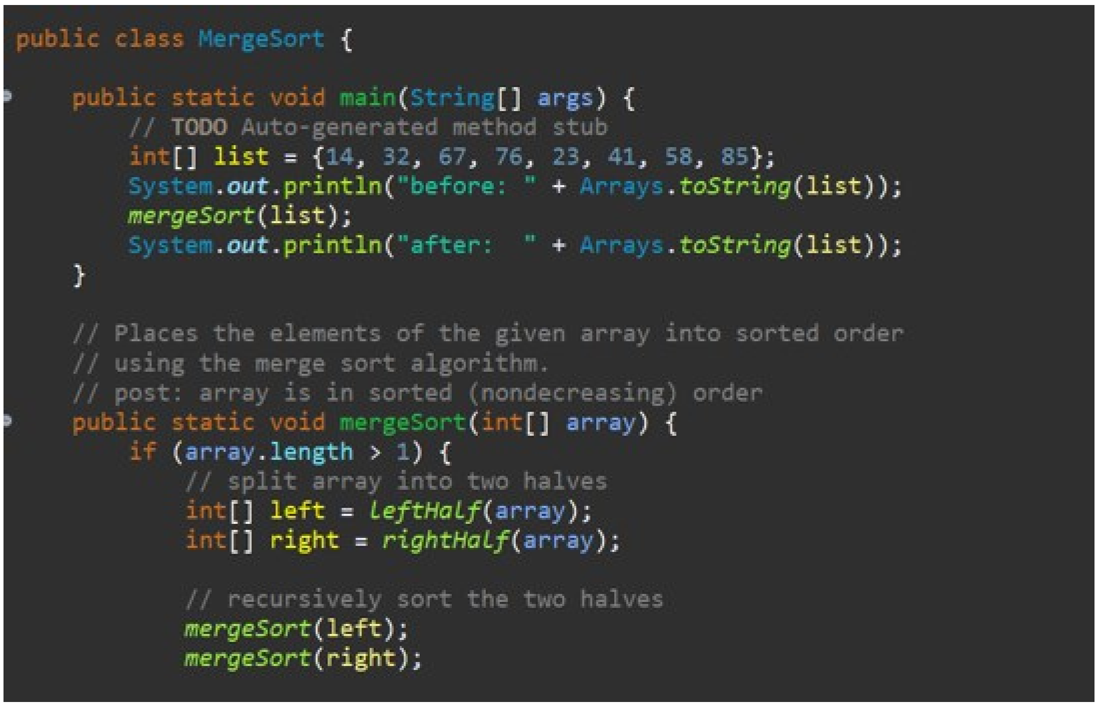
There are three main errors in the program:

- The leftHalf and rightHalf methods are called incorrectly; they should be passed the array without any modification.
- The merge method is incorrectly called with left++ and right--, which are invalid for array references; it should just be left and right.
- The result array is not initialized in the merge method, which will cause a NullPointerException.

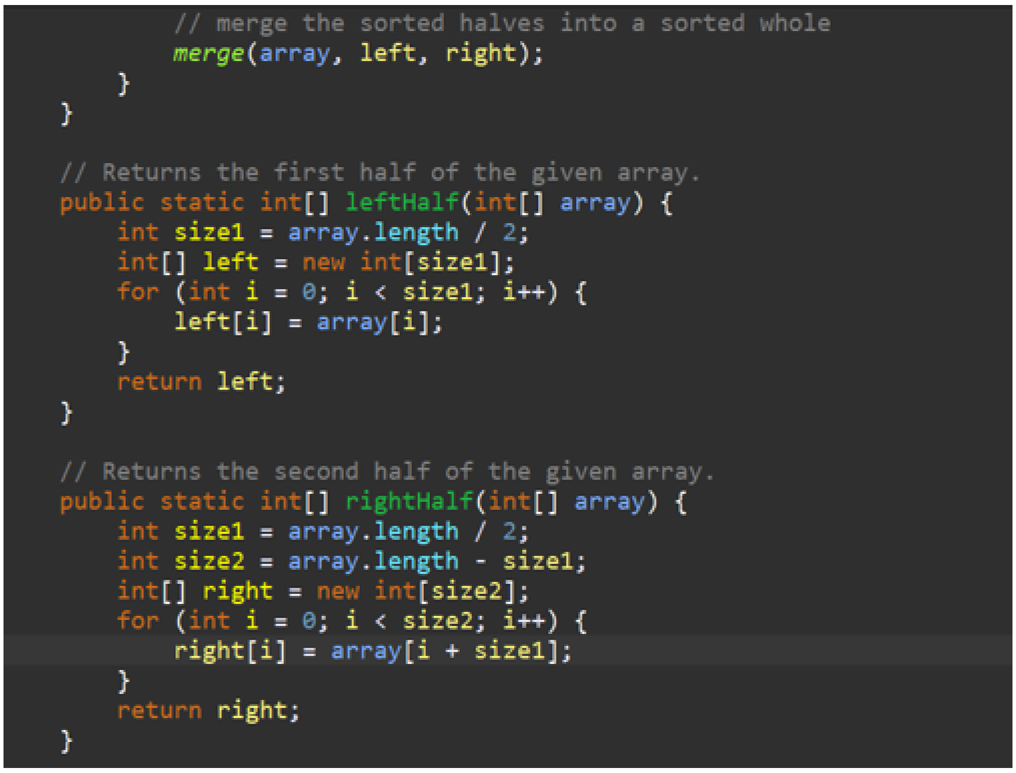
2. How many breakpoints you need to fix those errors? What are the steps you have taken to fix the error you identified in the code fragment?

Need breakpoints in the mergeSort, leftHalf, rightHalf, and merge methods to inspect how arrays are being passed and modified. To fix the errors, update the calls to leftHalf(array) and rightHalf(array), change the merge method call to merge(array, left, right), and initialize the result array in the merge method based on the combined lengths of left and right.

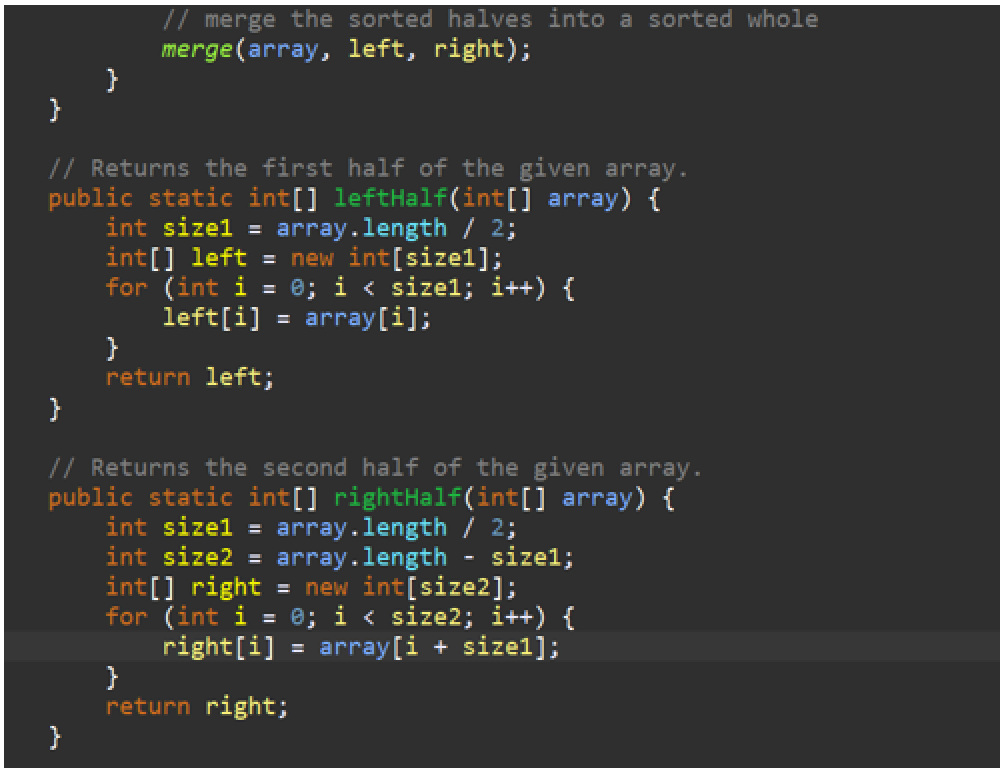
3. Submit your complete executable code?



```java
public class MergeSort {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        int[] list = {14, 32, 67, 76, 23, 41, 58, 85};
        System.out.println("before: " + Arrays.toString(list));
        mergeSort(list);
        System.out.println("after:  " + Arrays.toString(list));
    }

    // Places the elements of the given array into sorted order
    // using the merge sort algorithm.
    // post: array is in sorted (nondecreasing) order
    public static void mergeSort(int[] array) {
        if (array.length > 1) {
            // split array into two halves
            int[] left = leftHalf(array);
            int[] right = rightHalf(array);

            // recursively sort the two halves
            mergeSort(left);
            mergeSort(right);
```



```java
            // merge the sorted halves into a sorted whole
            merge(array, left, right);
        }
    }

    // Returns the first half of the given array.
    public static int[] leftHalf(int[] array) {
        int size1 = array.length / 2;
        int[] left = new int[size1];
        for (int i = 0; i < size1; i++) {
            left[i] = array[i];
        }
        return left;
    }

    // Returns the second half of the given array.
    public static int[] rightHalf(int[] array) {
        int size1 = array.length / 2;
        int size2 = array.length - size1;
        int[] right = new int[size2];
        for (int i = 0; i < size2; i++) {
            right[i] = array[i + size1];
        }
        return right;
    }
```



```
            // merge the sorted halves into a sorted whole
            merge(array, left, right);
        }
    }

    // Returns the first half of the given array.
    public static int[] leftHalf(int[] array) {
        int size1 = array.length / 2;
        int[] left = new int[size1];
        for (int i = 0; i < size1; i++) {
            left[i] = array[i];
        }
        return left;
    }

    // Returns the second half of the given array.
    public static int[] rightHalf(int[] array) {
        int size1 = array.length / 2;
        int size2 = array.length - size1;
        int[] right = new int[size2];
        for (int i = 0; i < size2; i++) {
            right[i] = array[i + size1];
        }
        return right;
    }
```

Code 7

1. How many errors are there in the program? Mention the errors you have identified.

There are several errors in the program. The first error is incorrect indexing in the matrix multiplication loop; it uses first[c-1][c-k] and second[k-1][k-d] instead of first[c][k] and second[k][d], leading to potential ArrayIndexOutOfBoundsException and incorrect results. The second error is an incorrect input prompt for the dimensions of the second matrix, which repeats the prompt for the first matrix. Lastly, the sum variable is not reset at the appropriate time, which can cause incorrect accumulation of values.

2. Which category of program inspection would you find more effective?

Code walkthroughs or peer reviews would be the most effective categories of program inspection for this program. These methods allow for collaborative examination of the code, where multiple perspectives can identify logical errors, especially in areas involving indexing and calculations.

3. Which type of error you are not able to identified using the program inspection?

Logical errors, such as those that arise from incorrect index access or miscalculations during matrix multiplication, may not be easily identified through program inspection alone. These errors may require running test cases to observe the actual output and validate the correctness of the algorithm.

4. Is the program inspection technique is worth applicable?

Yes, program inspection techniques are worth applying. They can help catch many types of errors early in the development process, improve code quality, and foster knowledge sharing among team members. However, they should be complemented with other testing methods, such as unit testing, to ensure comprehensive error detection.

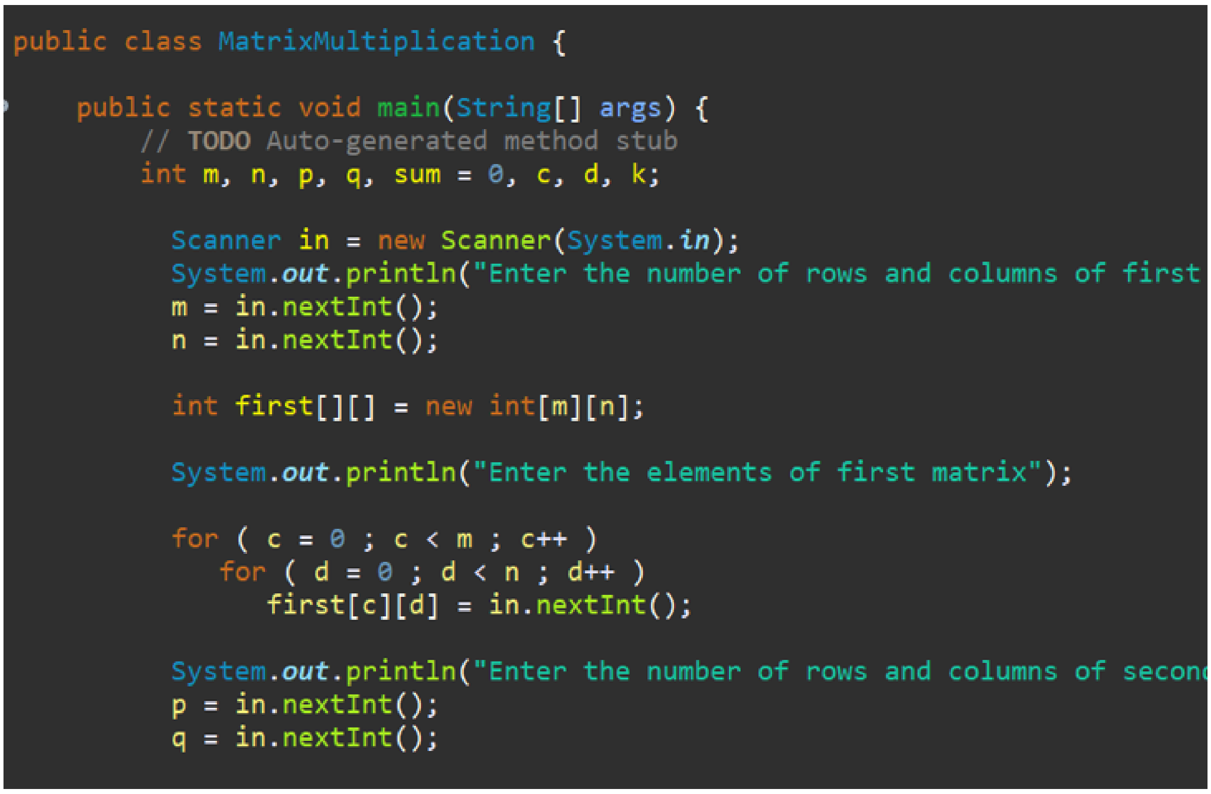
1. How many errors are there in the program? Mention the errors you have identified.

There are three main errors: incorrect matrix indexing in the multiplication loop (should use first[c][k] and second[k][d]), a repeated prompt for the second matrix dimensions instead of indicating the correct prompt, and the sum variable not being reset at the correct time.

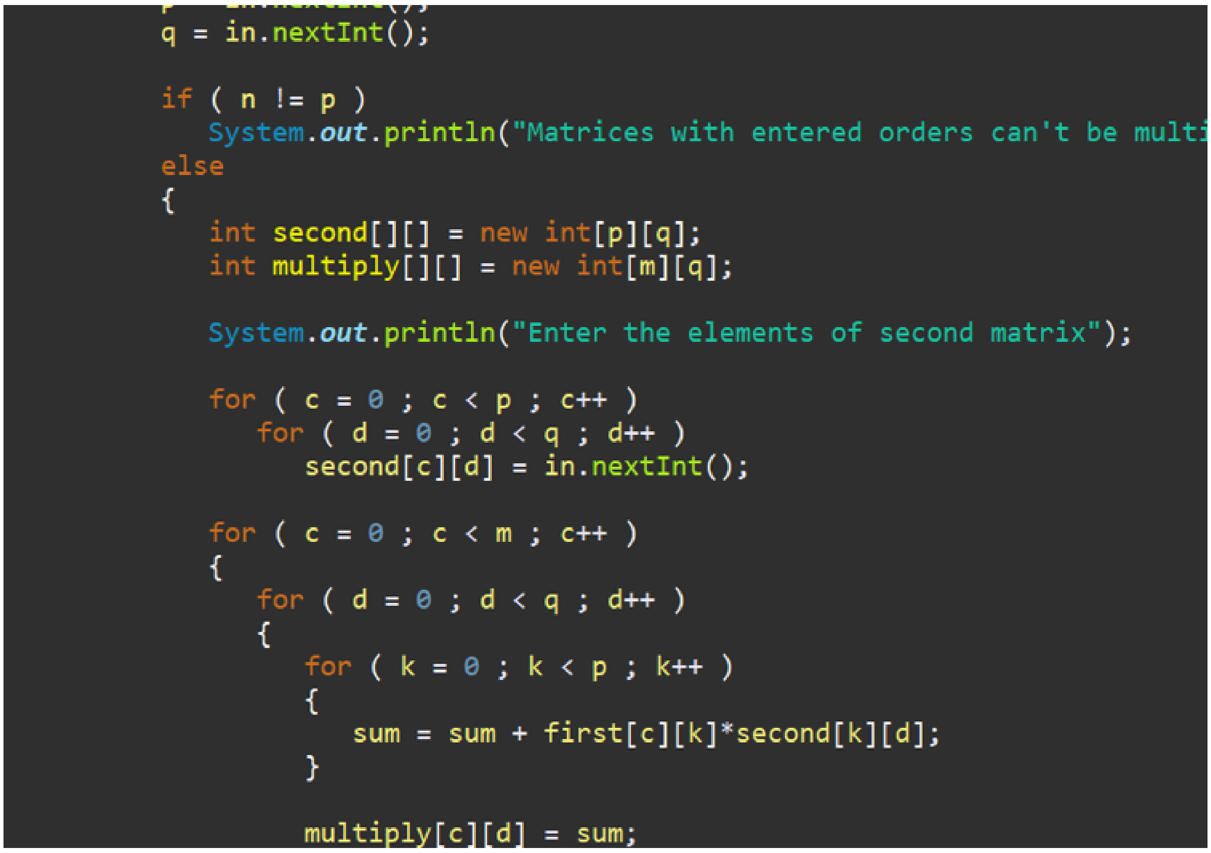
2. How many breakpoints you need to fix those errors? What are the steps you have taken to fix the error you identified in the code fragment?

would need breakpoints in the multiplication loop to inspect the values of first[c][k], second[k][d], and sum. To fix the errors, change the index access in the multiplication loop, update the prompts for the second matrix, and ensure sum is reset at the start of the innermost loop

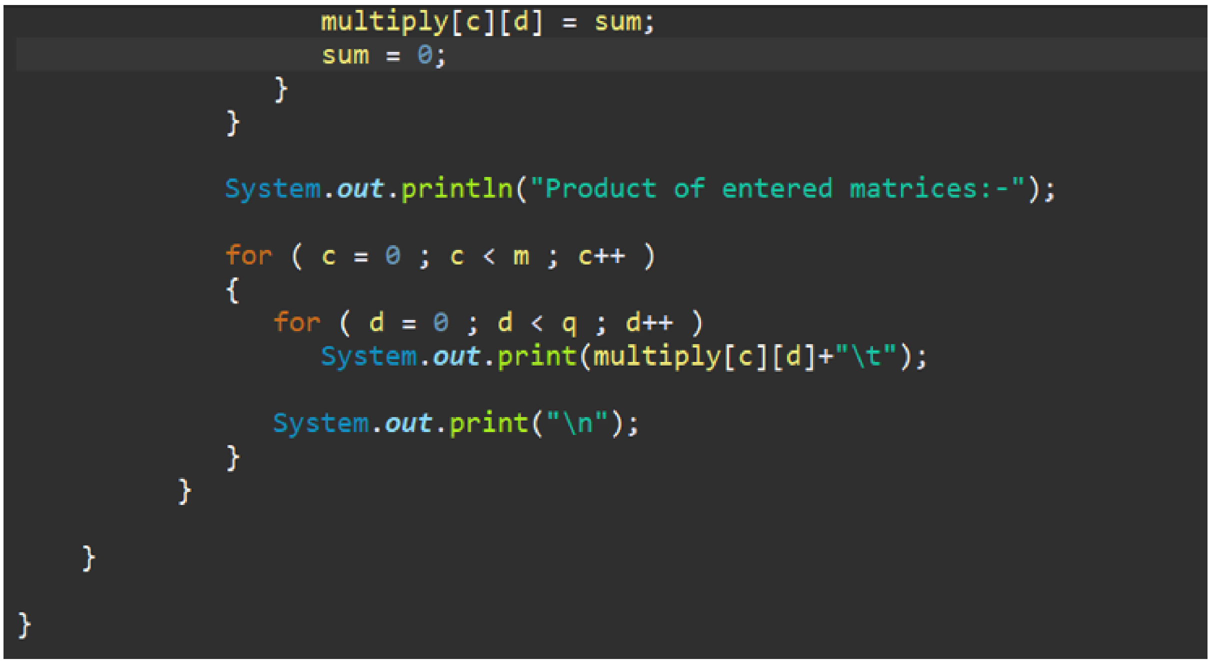
3. Submit your complete executable code?



```
public class MatrixMultiplication {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        int m, n, p, q, sum = 0, c, d, k;

          Scanner in = new Scanner(System.in);
          System.out.println("Enter the number of rows and columns of first
          m = in.nextInt();
          n = in.nextInt();

          int first[][] = new int[m][n];

          System.out.println("Enter the elements of first matrix");

          for ( c = 0 ; c < m ; c++ )
             for ( d = 0 ; d < n ; d++ )
                first[c][d] = in.nextInt();

          System.out.println("Enter the number of rows and columns of second
          p = in.nextInt();
          q = in.nextInt();
```



```
        q = in.nextInt();

        if ( n != p )
           System.out.println("Matrices with entered orders can't be multi
        else
        {
           int second[][] = new int[p][q];
           int multiply[][] = new int[m][q];

           System.out.println("Enter the elements of second matrix");

           for ( c = 0 ; c < p ; c++ )
              for ( d = 0 ; d < q ; d++ )
                 second[c][d] = in.nextInt();

           for ( c = 0 ; c < m ; c++ )
           {
              for ( d = 0 ; d < q ; d++ )
              {
                 for ( k = 0 ; k < p ; k++ )
                 {
                    sum = sum + first[c][k]*second[k][d];
                 }

                 multiply[c][d] = sum;
```



```
                 multiply[c][d] = sum;
                 sum = 0;
              }
           }

           System.out.println("Product of entered matrices:-");

           for ( c = 0 ; c < m ; c++ )
           {
              for ( d = 0 ; d < q ; d++ )
                 System.out.print(multiply[c][d]+"\t");

              System.out.print("\n");
           }
        }

    }

}
```

Code 9

1. How many errors are there in the program? Mention the errors you have identified.

There are four main errors identified in the program: the class name contains a space (Ascending _Order), the outer loop condition uses >= instead of <, the sorting comparison uses <= instead of >, and the output format incorrectly adds extra commas between elements.

2. Which category of program inspection would you find more effective?

Code reviews or pair programming would be the most effective categories of program inspection for this program, as they allow for collaborative examination and discussion of the logic, structure, and potential edge cases.

3. Which type of error you are not able to identified using the program inspection?

Logical errors, particularly in sorting logic or output formatting, may not be easily identified through program inspection alone. These types of errors often require testing with various input cases to uncover incorrect behavior.

4. Is the program inspection technique is worth applicable?

Yes, program inspection techniques are worth applying as they help in early detection of errors, improving code quality, and enhancing team collaboration. However, they should be complemented with additional testing methods, such as unit tests, to ensure comprehensive coverage of potential issues.

1. How many errors are there in the program? Mention the errors you have identified.

There are four main errors identified in the program:

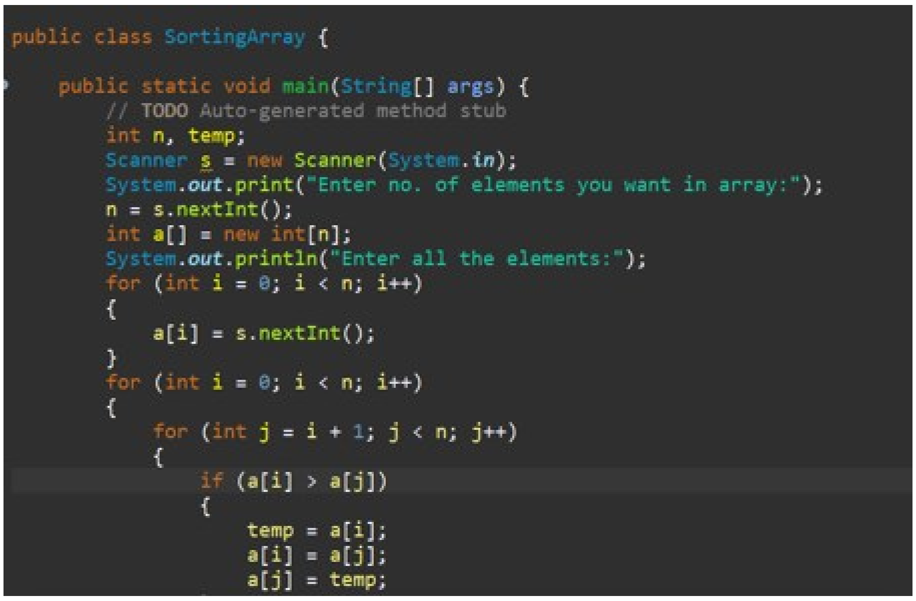
- The class name Ascending _Order contains a space, which is not allowed in Java.
- The outer loop condition for (int i = 0; i >= n; i++) should be changed to i < n to allow the loop to execute.
- The comparison in the sorting logic uses if (a[i] <= a[j]) instead of if (a[i] > a[j]), which is incorrect for ascending order.
- The output formatting has extra commas that should be avoided after the last element.

2. How many breakpoints you need to fix those errors? What are the steps you have taken to fix the error you identified in the code fragment?

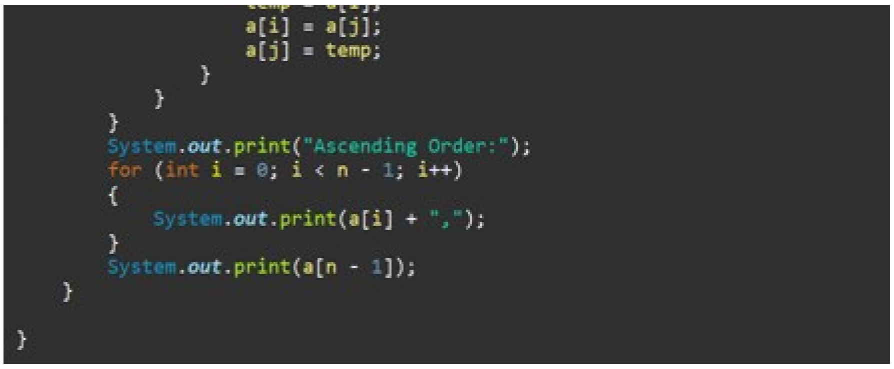
would need breakpoints in the sorting logic to inspect the values of i, j, and the elements in the array during the sorting process. To fix the errors:

- Rename the class to AscendingOrder.
- Change the outer loop condition from >= to <.
- Modify the comparison in the sorting logic from <= to >.
- Update the output formatting to prevent trailing commas.

3. Submit your complete executable code?



```
public class SortingArray {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        int n, temp;
        Scanner s = new Scanner(System.in);
        System.out.print("Enter no. of elements you want in array:");
        n = s.nextInt();
        int a[] = new int[n];
        System.out.println("Enter all the elements:");
        for (int i = 0; i < n; i++)
        {
            a[i] = s.nextInt();
        }
        for (int i = 0; i < n; i++)
        {
            for (int j = i + 1; j < n; j++)
            {
                if (a[i] > a[j])
                {
                    temp = a[i];
                    a[i] = a[j];
                    a[j] = temp;
```



```
                    a[i] = a[j];
                    a[j] = temp;
                }
            }
        }
        System.out.print("Ascending Order:");
        for (int i = 0; i < n - 1; i++)
        {
            System.out.print(a[i] + ",");
        }
        System.out.print(a[n - 1]);
    }

}
```

Code 10

1. How many errors are there in the program? Mention the errors you have identified.

There are several errors: the push method decrements top instead of incrementing it, the pop method incorrectly increments top when popping an element, the display method has a faulty loop condition (i > top should be i <= top), and the stack array is never updated after a pop operation.

2. Which category of program inspection would you find more effective?

Peer reviews would be the most effective inspection category, as they can help catch logical errors and issues with stack operations through discussion and code examination.

3. Which type of error you are not able to identified using the program inspection?

Logical errors related to the stack operations (like incorrect index management) might not be identified through inspection alone and may require testing to uncover.

4. Is the program inspection technique is worth applicable?

Yes, program inspection techniques are worth applying, as they help in catching errors early and improve code quality, but they should be complemented with testing to ensure full functionality.

1. How many errors are there in the program? Mention the errors you have identified.

There are five main errors identified in the program:

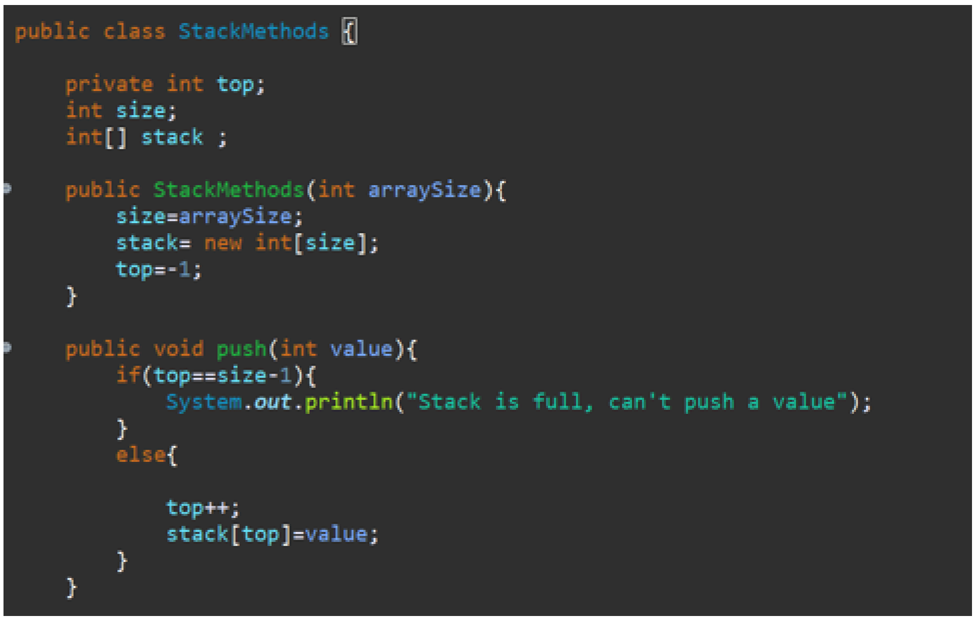
- In the push method, top is decremented instead of incremented when adding a value to the stack.
- In the pop method, top is incorrectly incremented when removing a value from the stack.
- The display method has a faulty loop condition (i > top should be i <= top).
- The pop method does not update the stack after popping an element (i.e., the element at the new top should be set to a default value like 0).
- The top should be initialized to -1, but the logic for pushing and popping is reversed.

2. How many breakpoints you need to fix those errors? What are the steps you have taken to fix the error you identified in the code fragment?

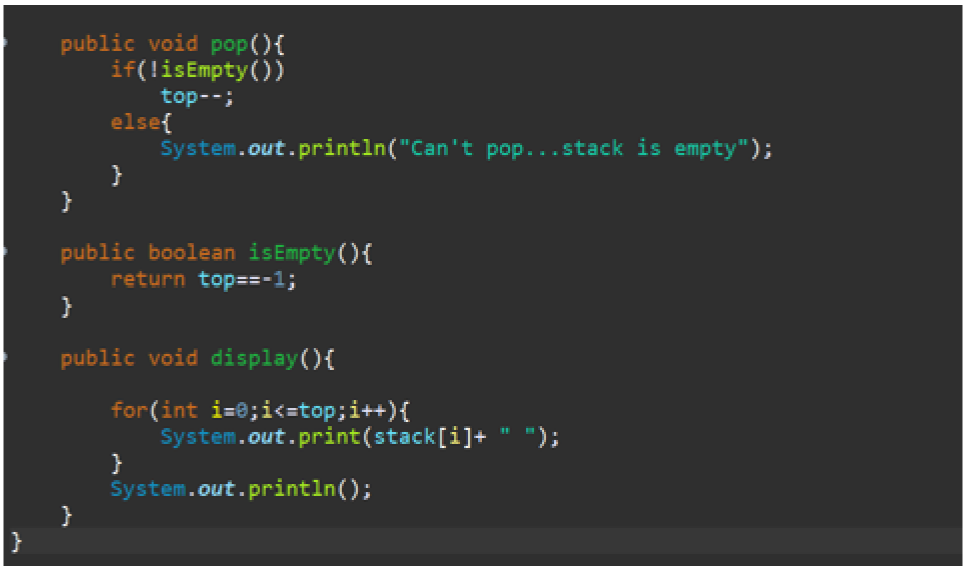
would need breakpoints in the push, pop, and display methods to inspect the values of top and the contents of the stack array during execution. Steps to fix the errors include:

- Change the push method to increment top when adding a new value.
- Modify the pop method to decrement top correctly.
- Update the loop condition in the display method to correctly iterate through the stack.
- In the pop method, set the popped element's position in the stack to a default value (optional for clarity).

3. Submit your complete executable code?



```
public class StackMethods {

    private int top;
    int size;
    int[] stack ;

    public StackMethods(int arraySize){
        size=arraySize;
        stack= new int[size];
        top=-1;
    }

    public void push(int value){
        if(top==size-1){
            System.out.println("Stack is full, can't push a value");
        }
        else{

            top++;
            stack[top]=value;
        }
    }
```



```java
    public void pop(){
        if(!isEmpty())
            top--;
        else{
            System.out.println("Can't pop...stack is empty");
        }
    }

    public boolean isEmpty(){
        return top==-1;
    }

    public void display(){

        for(int i=0;i<=top;i++){
            System.out.print(stack[i]+ " ");
        }
        System.out.println();
    }
}
```