# CorrectionModel Java Class Iterations

## Version A

```
package StemProject;
import java.util.Scanner;
public class RandomDotsManager {
   public static void main(String[] args) {
      // Create a scanner object to get user input
      Scanner scanner = new Scanner(System.in);
      // Get user input for rectangle dimensions
      System.out.print("Enter the width of the rectangle: ");
      int rectWidth = scanner.nextInt();
      scanner.nextLine();
      System.out.print("Enter the height of the rectangle: ");
      int rectHeight = scanner.nextInt();
      scanner.nextLine();
      // Get user input for the number of dots and the number of 2s and 3s
      System.out.print("Enter the total number of dots: ");
      int numDots = scanner.nextInt();
      scanner.nextLine();

      System.out.print("Enter the total number of 2s: ");
      int num2s = scanner.nextInt();
      scanner.nextLine();

      System.out.print("Enter the total number of 3s: ");
      int num3s = scanner.nextInt();
      scanner.nextLine();
      // Variables to track the best result (least number of sections)
      int minSections = Integer.MAX_VALUE;  // Initialize to a large value
      MapGenerator bestRun = null;
      // Infinite loop to keep running the program
      while (true) {
         MapGenerator randomDots = new MapGenerator(rectWidth, rectHeight, numDots, num2s, num3s);
         int sections = randomDots.getNumberOfDistricts();
         // Check if this run has the least sections so far
         if (sections < minSections) {
            minSections = sections;
            bestRun = randomDots;
            System.out.println("New best run with " + minSections + " sections (least so far)!");
```

```
            }
            // Ask the user whether they want to stop or continue
            System.out.print("Press Enter to stop or any other key to continue...");
            String input = scanner.nextLine();  // Read the user input
            if (input.isEmpty()) {
                break;  // Exit the loop if Enter is pressed
            }
        }
        // After stopping, print the best result (least number of sections)
        System.out.println("\nBest result with the least sections:");
        if (bestRun != null) {
            bestRun.countDotsInSections();  // Print the count of dots in sections for the best run
        }
        // Close the scanner
        scanner.close();
    }
}
```

# Version B

```
package StemProject;
import java.util.Scanner;
public class CorrectionModel {
    public static void main(String[] args) {
        // Create a scanner object to get user input
        Scanner scanner = new Scanner(System.in);
        // Get user input for region dimensions
        System.out.print("Enter the width of the region: ");
        int regionWidth = scanner.nextInt();
        scanner.nextLine();

        System.out.print("Enter the length of the region: ");
        int regionHeight = scanner.nextInt();
        scanner.nextLine();

        // Get user input for the number of voters and the number of Party A voters (1) and Party B voters (2)
        System.out.print("Enter the total number of voters: ");
        int numVoters = scanner.nextInt();
        scanner.nextLine();

        System.out.print("Enter the total number of Party A voters: ");
        int num1s = scanner.nextInt();
        scanner.nextLine();
```

```
        System.out.print("Enter the total number of Party B voters: ");
        int num2s = scanner.nextInt();
        scanner.nextLine();
        // Variables to track the best result (least number of sections)
        int minDistricts = Integer.MAX_VALUE;  // Initialize to a larger value
        MapGenerator bestRun = null; //Start at null
        // Infinite loop to keep running the program
        while (true) {
            MapGenerator randomVoters = new MapGenerator(regionWidth, regionHeight, numVoters, num1s, num2s);
            int districts = randomVoters.getNumberOfDistricts();

            // Check if this run has the least sections so far
            if (districts < minDistricts) {
                minDistricts = districts;
                bestRun = randomVoters;
                System.out.println("New best run with " + minDistricts + " districts (least so far)!");
            }
            // Ask the user whether they want to stop or continue
            System.out.print("Press Enter to stop or any other key to continue...");
            String input = scanner.nextLine();  // Read the user input
            if (input.isEmpty()) {
                break;  // Exit the loop if Enter is pressed
            }
        }
        // After stopping, print the best result (least number of districts)
        System.out.println("\nBest result with the least districts:");
        if (bestRun != null) {
            bestRun.countVotersInDistricts();  // Print the count of voters in districts for the best run
        }
        // Close the scanner
        scanner.close();
    }
}
```

# Version C

```
package StemProject;
import java.util.Scanner;
public class CorrectionModel {
    public static void main(String[] args) {
        // Create a scanner object to get user input
        Scanner scanner = new Scanner(System.in);
        // Get user input for region dimensions
        System.out.print("Enter the width of the region: ");
        int regionWidth = scanner.nextInt();
```

```
scanner.nextLine();

System.out.print("Enter the length of the region: ");
int regionHeight = scanner.nextInt();
scanner.nextLine();

// Get user input for the number of voters and the number of Party A voters (1) and Party B voters (2)
System.out.print("Enter the total number of voters: ");
int numVoters = scanner.nextInt();
scanner.nextLine();

System.out.print("Enter the total number of Party A voters: ");
int num1s = scanner.nextInt();
scanner.nextLine();

System.out.print("Enter the total number of Party B voters: ");
int num2s = scanner.nextInt();
scanner.nextLine();

// Variables to track the best result (least number of sections)
int minDistricts = Integer.MAX_VALUE;  // Initialize to a large value
MapGenerator bestRun = null;
// Infinite loop to keep running the program
while (true) {
    MapGenerator randomVoters = new MapGenerator(regionWidth, regionHeight, numVoters, num1s, num2s);
    int districts = randomVoters.applyIdentificatioModel();

    // Check if this run has the least sections so far
    if (districts < minDistricts) {
        minDistricts = districts;
        bestRun = randomVoters;
        System.out.println("New best run with " + minDistricts + " districts (least so far)!");
    }
    // Ask the user whether they want to stop or continue
    System.out.print("Press Enter to stop or any other key to continue...");
    String input = scanner.nextLine();  // Read the user input
    if (input.isEmpty()) {
        break;  // Exit the loop if Enter is pressed
    }
}
// After stopping, print the best result (least number of districts)
System.out.println("\nBest result with the least districts:");
if (bestRun != null) {
    bestRun.applyIdentificatioModel();  // Print the count of voters in districts for the best run
}
// Close the scanner
scanner.close();
```

```
    }
}
```

# Version D

```
package StemProject;

import java.util.Scanner;

public class CorrectionModel {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Get user input for region dimensions
        System.out.print("Enter the width of the region: ");
        int regionWidth = scanner.nextInt();
        scanner.nextLine();

        System.out.print("Enter the length of the region: ");
        int regionHeight = scanner.nextInt();
        scanner.nextLine();

        // Get user input for the number of voters and the number of Party A and Party B voters
        System.out.print("Enter the total number of voters: ");
        int numVoters = scanner.nextInt();
        scanner.nextLine();

        System.out.print("Enter the total number of Party A voters: ");
        int numAs = scanner.nextInt();
        scanner.nextLine();

        System.out.print("Enter the total number of Party B voters: ");
        int numBs = scanner.nextInt();
        scanner.nextLine();

        // Variables to track the best result (lowest gerrymandering score)
        double bestGerrymanderingScore = Double.MAX_VALUE; // Initialize to a large value
        MapGenerator bestRun = null;

        // Infinite loop to keep running the program
        while (true) {
            // Create a new random map with the provided dimensions and voter counts
            MapGenerator randomVoters = new MapGenerator(regionWidth, regionHeight, numVoters, numAs, numBs);

            // Calculate the gerrymandering score for this run
```

```
            double gerrymanderingScore = applyIdentificationModel(randomVoters);

            // Check if this run has the best (lowest) gerrymandering score so far
            if (gerrymanderingScore < bestGerrymanderingScore) {
                bestGerrymanderingScore = gerrymanderingScore;
                bestRun = randomVoters;
                System.out.println("New best run with gerrymandering score: " + bestGerrymanderingScore);
            }

            // Ask the user whether they want to stop or continue
            System.out.print("Press Enter to stop or any other key to continue...");
            String input = scanner.nextLine(); // Read the user input

            if (input.isEmpty()) {
                break; // Exit the loop if Enter is pressed
            }
        }

        // After stopping, print the best result (lowest gerrymandering score)
        System.out.println("\nBest result with the lowest gerrymandering score:");
        if (bestRun != null) {
            System.out.println("Gerrymandering Score: " + bestGerrymanderingScore);
            // You can display additional information if needed, like the district breakdown, by calling bestRun methods
        }

        // Close the scanner
        scanner.close();
    }

    // Method to calculate the gerrymandering score / apply the identification model (V4)
    public static double applyIdentificationModel(MapGenerator map) {
        // Extract metrics from the map
        double effGap = Math.abs(
            (map.getLosing1Votes() + map.getExcessWinning1Votes()) -
            (map.getLosing2Votes() + map.getExcessWinning2Votes())
        ) / (double) map.getTotalVotes();

        double pvd1 = Math.abs(
            (double) map.getTotal1Votes() / map.getTotalVotes() -
            (double) map.getDistricts1Won() / map.getTotalDistricts()
        );

        double pvd2 = Math.abs(
            (double) map.getTotal2Votes() / map.getTotalVotes() -
            (double) map.getDistricts2Won() / map.getTotalDistricts()
        );
```

```
        double identificationModelScore = (effGap * 0.5) + (pvd1 * 0.25) + (pvd2 * 0.25);

        // Calculate the final score (this is just one possible way to combine metrics)
        return identificationModelScore;
    }
}
```