LIVERPOOL JOHN MOORES UNIVERSITY

BEACONHOUSE INTERNATIONAL COLLEGE

Coursework 1: A Police Stop and Search Data Analysis System

Data Structures and Algorithms

Team members

Haram Sabir
Aimal Sadia Khan
Vaneeza
Sara

A Police Stop and Search Data Analysis System

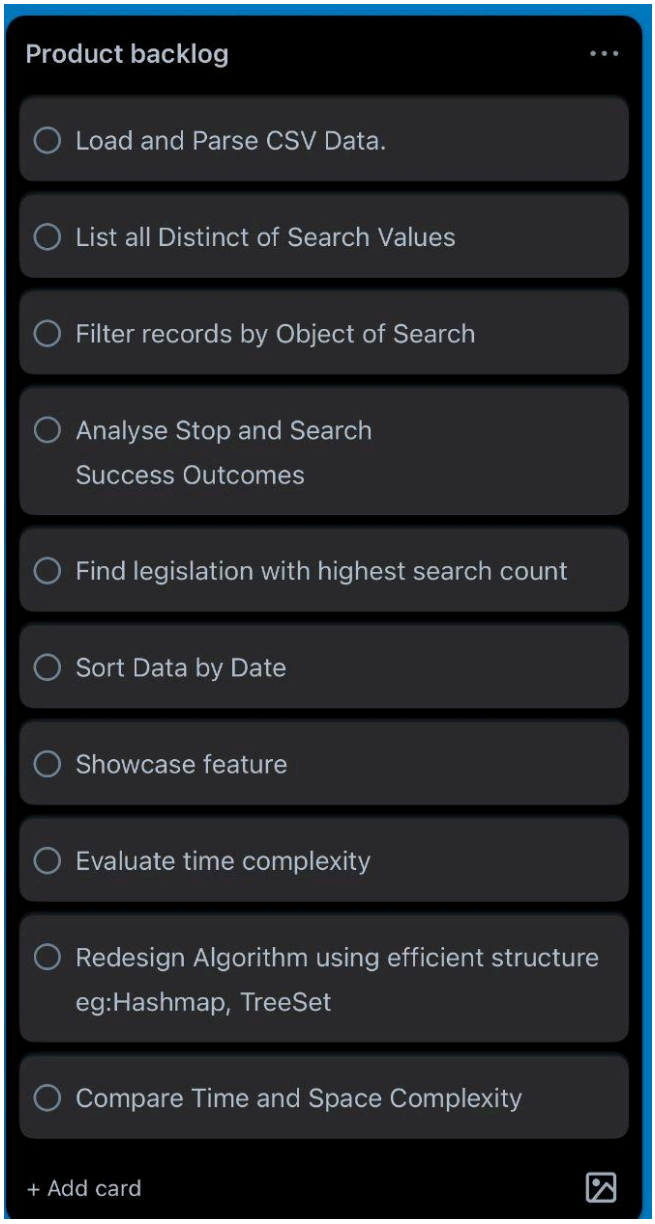
1. Introduction

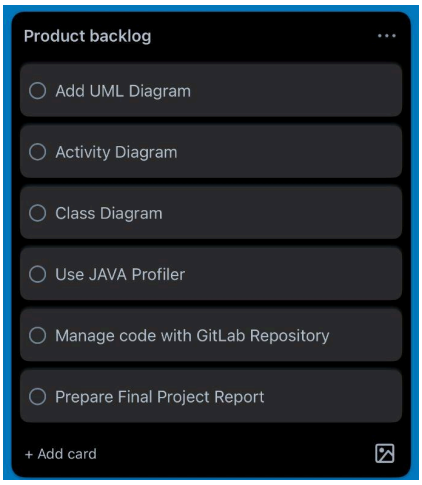
The primary goal of our project is to analyze, design, and implement a Java console-based application that reads, processes, and stores real-world stop, search, and police records datasets. The main goal of our project is to analyze, design and implement a java console based application that reads, processes and stores real-world police stop-search-record datasets. We wanted to understand how data can be processed more efficiently and how algorithm design affects performance. This system allows users to upload multiple CSV files from different regions like Cheshire and Mereyside. Load multiple CSV files from different regions like Cheshire and Mereyside.

2. Project and Team Work

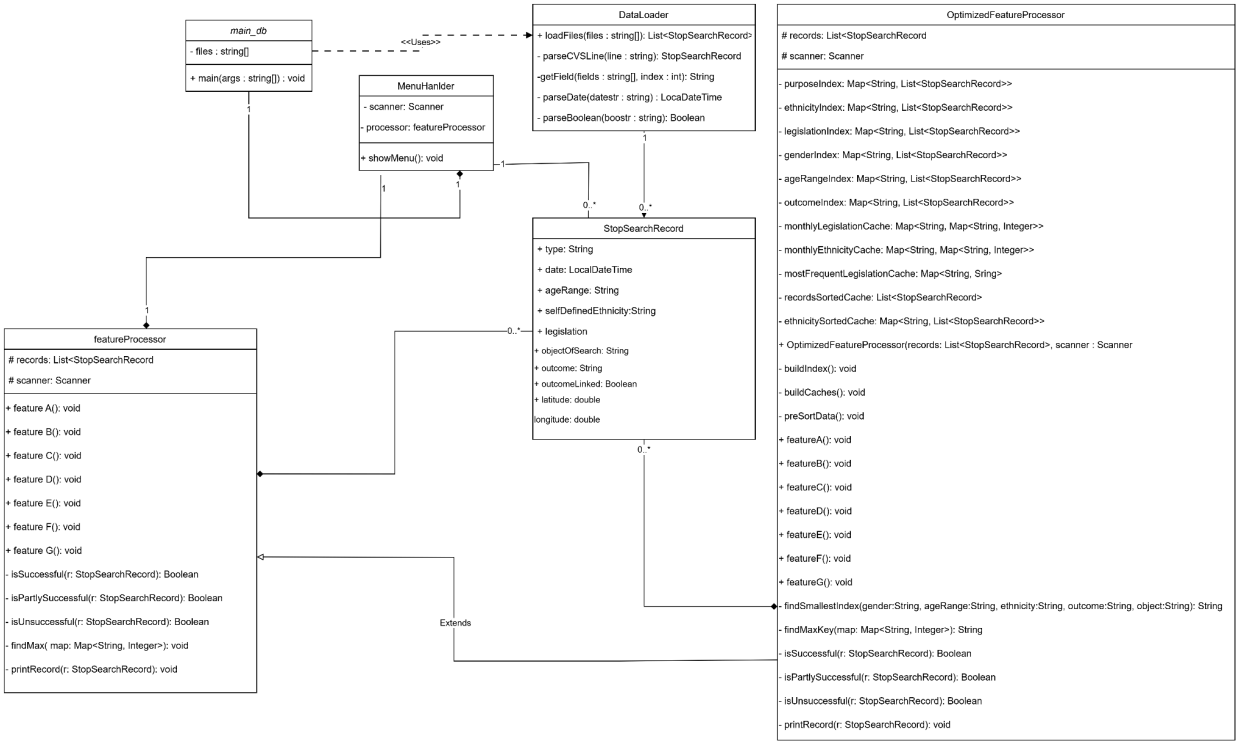
For this project, our group worked together to plan, manage, and track all tasks using Trello. We created a Trello board with different lists, including Product Backlog, In Progress, Testing, and Completed, to help us organize work clearly. Done, to help us organize work clearly.

3. Product Backlog





4. Class Diagram



[PoliceDataBase.drawio]

The purpose of the class diagram is to give a clear overview of how the program is structured.

- Main_db is the main entry of the program. It controls the flow of execution and connects all other classes. the other classes
- DataLoader loads and analyzes CSV files containing police stop and search data
- FeatureProcessor analyzes and processes the data using several features. parse the CSV files containing police
- Stop-and-search datafeatureProcessor analyzes and processes data using multiple features.
- OptimizedFeatureProcessor improves data processing performance using indexes, caches and pre-sorted lists. the performance of data processing using indexes, caches, and pre-sorted lists.

Relationships:

➢ Main_db → DataLoader:

- Type: Dependency(uses)
- Main_db depends on the dataloader to load data from CSV files. It doesn't store a DataLoader object permanently, instead just creates it temporarily and call its methods.

➢ Main_db → MenuHandler:

- Type: Composition(◇))
- Main_db creates MenuHandler

➢ DataLoader → StopSearchRecord:

- Type: Association(1 → 0..*)
- Each DataLoader creates many StopSearchRecord objects while parsing CSV files.

➢ FeatureProcessor → StopSearchRecord:

- Type: Composition(◇)
- FeatureProcessor owns the list of StopSearchRecord objects it analyzes.

➢ MenuHandler → FeatureProcessor

- Type: Composition(◇)
- MenuHandler contains or owns a reference to a FeatureProcessor object.

➢ MenuHandler → StopSearchRecord:

- Type: Association(indirect)
- MenuHandler doesn't store records itself but passes them indirectly to FeatureProcessor.

➢ FeatureProcessor → OptimizedFeatureProcessor:

- Type: Inheritance(Generalization)
- OptimizedFeatureProcessor inherits from FeatureProcessor. It reuses its logic and adds more efficient data structures like caches and indexes.

➢ OptimizedFeatureProcessor → StopSearchRecord:

- Type: Composition(◇)
- The optimized version also owns a list of StopSearchRecord objects and several maps(indexes) that reference them.

5. System Design

Our system is designed modularly, which means that each part has its own task. in a modular way, which means each part has its own job. This makes the code easier to read, fix and improve in the future (for example, if we want to add a database or a suitable interface later). fix, and improve in the future (for example, if we want to add a database or a proper interface later).

5.1 System Architecture

The whole program is divided into three main layers

Data Layer: This part is responsible for reading CSV files. the CSV files. It also does basic configuration and indexing that makes the optimized version easier. some basic setup and indexing that helps in the optimised version.

Processing Layer: This layer contains the main logic for all features of the base (O(N)) and faster (O(1)) versions. features of both the basic (O(N)) and the faster (O(1)) versions. It manages search, counting and sorting operations. handles searching, counting, and sorting operations.

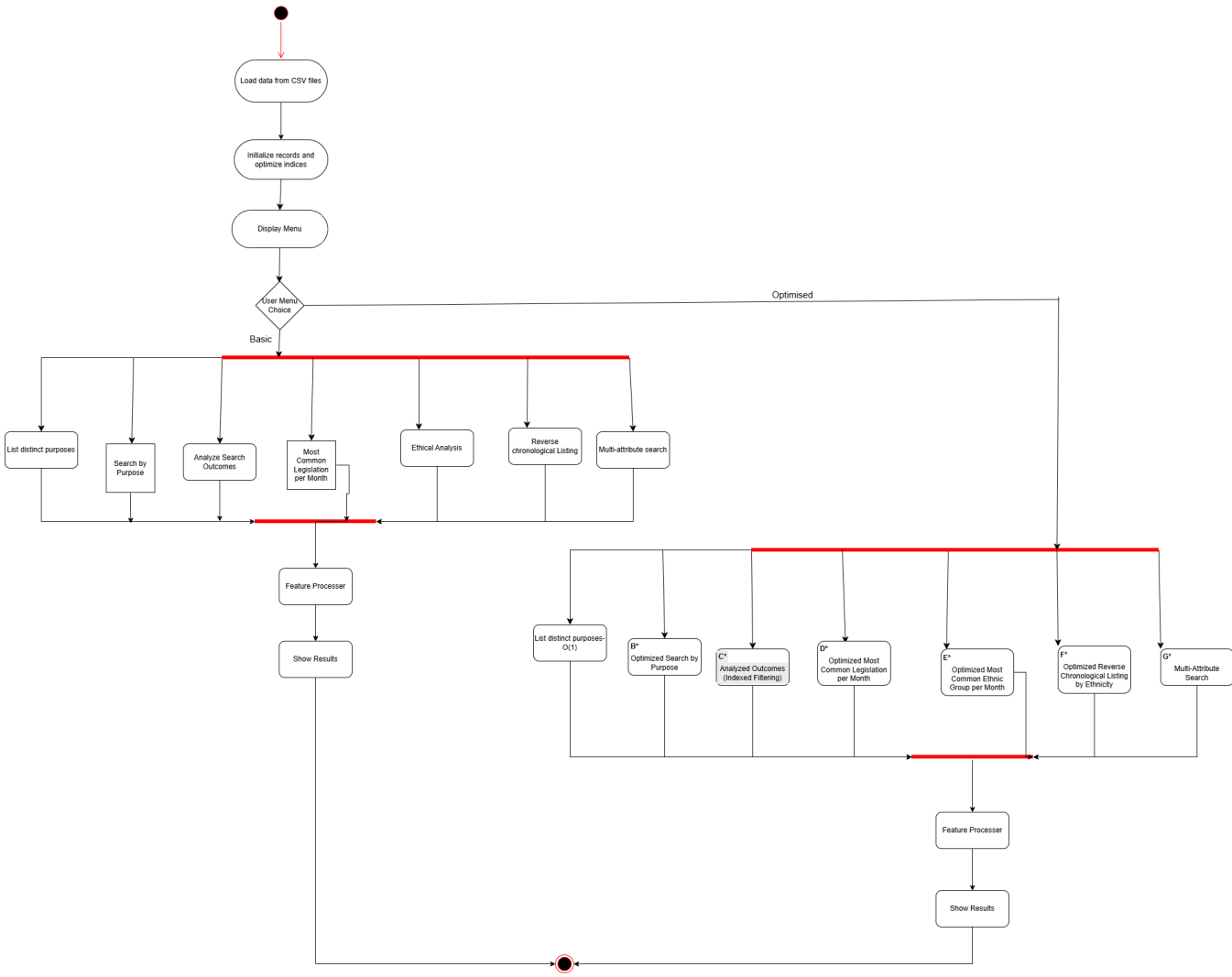
Interface Layer: This is the text menu that the user interacts with. text-based menu the user interacts with. It allows the user to choose different features and easily see the results. lets the user choose different features and see results easily.

5.2 Main Components

main_db class: Handles loading data, displaying the menu, taking user input into account, and executing the selected functionality. showing the menu, taking user input, and running the selected feature. It also manages all the indexes of the optimized version. for the optimised version

- .**StopSearchRecord class:** Stores a record of the dataset, including date, gender, age, ethnicity, legislation, search result and purpose. one record from the dataset, including date, gender, age, ethnicity, legislation, outcome, and search purpose.
- **Index Structures:** Used to speed up searches and analyses. make searches and analysis faster.
- **purposeIndex:** Quick search by purpose.
- **ethnicIndexSorted:** Ethnic data Sorted for quick searching. ethnicity data for fast lookup.
- **monthlyLegislationCounts / monthlyEthnicCounts:** Count and group by month.
- **monthlyMaxLegislation / monthlyMaxEthnicGroup:** Store monthly best results for instant output. top monthly results for instant output.

6. Activity Diagram



[Updated Activity (9).drawio]

The activity diagram shows how the Police Stop and Search Analysis System works, step by step:

- The system starts by loading the CSV data into memory. works step by step:
- The system starts by loading the CSV data into memory.
- Every record is read and savedIndexes are created to make optimized features work faster.
- Each record is read and savedIndexes are created to make the optimised features work faster.
- The user sees a menu with all features (A–G)

- . Once the user selects a feature, the system executes it using the basic (O(N)) or optimized (O(1)) method.
- After the user selects a feature, the system runs it using either the basic (O(N)) or optimised (O(1)) method.

7.Main Algorithm

7.1 Data Loading

The program starts by extracting all data from a set of CSV files. kicks off by pulling all the data from a set of CSV files. Each file lists details, such as the date, gender, age range, what was searched, current legislation, the result and the ethnicity of the subject. like the date, gender, age bracket, what was searched for, the governing legislation, the outcome, and the subject's ethnicity. It then reads each line, turning each line into a record added to a list. every line, turning each row into a record that gets appended to a list. This list of records becomes the backbone of the system's search and analysis. That list of records becomes the backbone for the system's searching and analysis.

7.2 Menu and System Versions

Once the data loading is complete, the application displays its menu allowing the user to select one of the two versions of the system

Basic version: It simply scans the set record by record using a linear search.
Optimized version: It uses indexing and caching to make searches happen faster. taps into indexing and caching so the searches zip along faster.

7.3 Main Menu and Options

When the user selects a version, the application displays a menu listing options A through G, reserving X for exit. throws up a menu listing options A to G, reserving X for exiting. Each input triggers an operation. Every entry triggers an operation. When an option is selected, the program relies on a switch statement to determine which feature was chosen, then triggers that part of the code. leans on a switch statement to figure out which feature was chosen and then fires off that portion of the code.

Features

- **Feature A:** Gathers a list of all objects searched by users. Aggregates a list of every object that users have searched for.
- **Feature B:** Prompts the user to enter a search object, then displays all searches related to it. type in a search object, then surfaces all the searches that relate to it.
- Feature C: Reviews the results and records the number of successes, partial successes, and failures. outcomes and records the count of successes, partial successes, and failures.
- **Feature D:** Ask for a month, then display frequently referenced and most successful legislation in that month. Asks for a month, then displays the legislation that was frequently referenced and the one that attained the greatest success during that month.
- **Feature E:** Analyzes data by self-defined ethnicity and highlights the group that saved the searches. Parses the data by self-defined ethnicity and highlights the group that logged the searches.
- **Feature F:** Retrieves all entries related to the selected ethnicity and orders them based on date. Pulls up every entry linked to the selected ethnicity and orders them from the date backward.
- **Feature G:** Allows the user to provide a handful of details about gender, age range, applicable legislation and search purpose to bring up the necessary results. Lets the user feed in a handful of details gender, age range, applicable legislation, and the object of the search to surface the results needed.

  After a feature wraps up, the application asks the user whether to return to the menu or exit. This prompt repeats continuously until the exit option is finally chosen.

7.4 Optimized Version Explanation

The optimized version speeds things up by creating indexes and caching data. constructing indexes and caching data. In practice, this means that it groups the information into categories: legislation, ethnicity, research object so that the program can access the slice directly instead of rescanning each record. means it bins the information into categories legislation, ethnicity, search object so the program can jump straight to the slice instead of scanning every record again. In contrast, the basic version simply iterates through the list record By record. In contrast, the basic version simply walks through the list record by record.

7.5 How It Works Overall

When the curtain falls, the algorithm rises, transforming a cumbersome tool into one that users can actually navigate without hassle. It's the algorithm that does the lifting, turning a clunky tool into something users can actually navigate without a headache. It retrieves each data file on launch, offers a set of options to the user, initiates the search or analysis, and then returns the response in a clear, readable format. hoovers up every data file at launch, lays a set of options before the user, fires off the search or analysis, and then flashes the answer back in a crisp, readable format. After that, the program continues to run, buzzing contentedly until the user finally presses the button. just glides along, humming contentedly until the user finally hits the button.

Feature Summary

- Feature D: Request a month and display the most used and successful legislation in that month. Asks for a month and shows the most used and most successful legislation in that month.
- Feature E: Analyzes data by self-defined ethnicity and shows which group has been searched for the most. Analyses the data by self-defined ethnicity and shows which group had the most searches.
- Feature F: Lists all records for a chosen ethnicity in reverse order by date. the records for a chosen ethnicity in reverse order by date.
- Feature G: Allows the user to enter multiple details such as gender, age range, legislation and search subject to find specific results. like gender, age range, legislation, and object of search to find specific results.

7.6 Program Flow

After completing a feature, the program asks if the user wants to return to the main menu or exit. any feature, the program asks if the user wants to go back to the main menu or exit. This process continues to repeat until the user selects the exit option. keeps repeating until the user selects the exit option.

7.7 Algorithm Efficiency

The optimized version works faster because it creates indexes and caches. builds indexes and caches. This means that it groups data by categories such as legislation, ethnicity and search subject, so the program can find items quickly without re-checking

each record. like legislation, ethnicity, and object of search, so the program can find things quickly without checking every record again. The basic version, on the other hand, checks each record in the list one by one.

Ultimately, the algorithm makes the program easy to use. In the end, the algorithm makes the program easy to use. It loads all the data at the beginning, allows the user to choose what they want to do, performs the correct search or analysis and displays the result clearly. start, lets the user pick what they want to do, performs the right search or analysis, and displays the result clearly. The program continues to run correctly until the user chooses to exit. keeps running smoothly until the user chooses to exit.

8 Implementation

The system is built in using Java.util,java.io

8.1 Basic Features (O(N))

- A**:** Show all the different research objectives. search purposes.
- B: Objective Search using normal linear search. by purpose using normal linear search.
- C: Check the search results (successful, partially successful, failed). outcomes (successful, partly successful, unsuccessful)
- D: Find the most current legislation each month. common legislation each month. E: Check the most common ethnic group by month or legislation.
- F: Show records in reverse order by date.
- G: Search using multiple filters at once. more than one filter at a time.

8.2 Optimised Features (O(1))

- B*: Quick search by objective using the index. Fast search by purpose using index.
- D*: Monthly analysis of legislation based on recorded results. analysis using saved results.
- E*: Analysis of Ethnic groups using pre-calculated results. group analysis using pre-calculated results.
- F*: Uses already sorted ethnicity data for faster program listing. Loop faster listing.

Code

Class Main_db:

```
package main_db;
import java.util.*;
public class main_db {

  public static void main(String[] args) {
    String[] files = {
      "C:\\Users\\DELL\\git\\a-police-data-analysis-system\\database\\src\\main\\java\\main_db\\2025-06-cheshire-stop-and-search.csv",
      "C:\\Users\\DELL\\git\\a-police-data-analysis-system\\database\\src\\main\\java\\main_db\\2025-07-cheshire-stop-and-search.csv",
      "C:\\Users\\DELL\\git\\a-police-data-analysis-system\\database\\src\\main\\java\\main_db\\2025-08-cheshire-stop-and-search.csv",
      "C:\\Users\\DELL\\git\\a-police-data-analysis-system\\database\\src\\main\\java\\main_db\\2025-09-cheshire-stop-and-search.csv",
      "C:\\Users\\DELL\\git\\a-police-data-analysis-system\\database\\src\\main\\java\\main_db\\2025-06-merseyside-stop-and-search.csv",
      "C:\\Users\\DELL\\git\\a-police-data-analysis-system\\database\\src\\main\\java\\main_db\\2025-07-merseyside-stop-and-search.csv",
      "C:\\Users\\DELL\\git\\a-police-data-analysis-system\\database\\src\\main\\java\\main_db\\2025-08-merseyside-stop-and-search.csv",
      "C:\\Users\\DELL\\git\\a-police-data-analysis-system\\database\\src\\main\\java\\main_db\\2025-09-merseyside-stop-and-search.csv",
    };
    DataLoader loader = new DataLoader();
    List<StopSearchRecord> records = loader.loadFiles(files);
    Scanner scanner = new Scanner(System.in);

    System.out.println("Choose mode:");
    System.out.println("1. Normal Application");
    System.out.println("2. NetBeans Profiling Mode");
    System.out.print("Choice: ");

    String choice = scanner.nextLine();

    if (choice.equals("2")) {
      profileAllFeatures(records);
    } else {
      MenuHandler menu = new MenuHandler(records, scanner);
      menu.showMenu();
    }

    scanner.close();
  }

  public static void profileAllFeatures(List<StopSearchRecord> records) {
    System.out.println("=== NETBEANS PROFILING MODE ===");
    System.out.println("Dataset size: " + records.size() + " records");
    System.out.println("Starting performance profiling...\n");

    FeatureProcessor basic = new FeatureProcessor(records, new Scanner(System.in));
    OptimizedFeatureProcessor optimized = new OptimizedFeatureProcessor(records, new Scanner(System.in));

    System.out.println("🔄 Running Feature A (Distinct Purposes)...");
    for (int i = 0; i < 100; i++) {
      basic.featureA();
      optimized.featureA();
    }

    System.out.println("🔄 Running Feature B (Search by Purpose)...");
    for (int i = 0; i < 50; i++) {
      simulateFeatureB(basic, optimized, "Controlled drugs");
      simulateFeatureB(basic, optimized, "Offensive weapons");
    }

    System.out.println("🔄 Running Feature C (Outcome Analysis)...");
    for (int i = 0; i < 80; i++) {
      basic.featureC();
      optimized.featureC();
    }
```

```
        System.out.println("🔄 Running Feature D (Legislation by Month)...");
        for (int i = 0; i < 60; i++) {
            simulateFeatureD(basic, optimized);
        }

        System.out.println("🔄 Running Feature F (Reverse Chronological)...");
        for (int i = 0; i < 40; i++) {
            simulateFeatureF(basic, optimized);
        }

        System.out.println("✅ PROFILING COMPLETE - Check NetBeans Profiler results!");
    }

    private static void simulateFeatureB(FeatureProcessor basic, OptimizedFeatureProcessor optimized, String purpose) {
        int count = 0;
        for (StopSearchRecord r : basic.records) {
            if (r.objectOfSearch.equalsIgnoreCase(purpose)) count++;
        }
        List<StopSearchRecord> results = optimized.getPurposeIndex().getOrDefault(purpose.toLowerCase(), new ArrayList<>());
    }
    private static void simulateFeatureD(FeatureProcessor basic, OptimizedFeatureProcessor optimized) {
        Map<String, Integer> freq = new HashMap<>();
        for (StopSearchRecord r : basic.records) {
            if (r.date != null && r.date.getMonthValue() == 6 && !r.legislation.isEmpty()) {
                freq.merge(r.legislation, 1, Integer::sum);
            }
        }

        Map<String, Integer> cached = optimized.getMonthlyLegislationCache().getOrDefault("6-2025", new HashMap<>());
    }
    private static void simulateFeatureF(FeatureProcessor basic, OptimizedFeatureProcessor optimized) {
        List<StopSearchRecord> basicResults = new ArrayList<>();
        for (StopSearchRecord r : basic.records) {
            if (r.selfDefinedEthnicity.toLowerCase().contains("white")) {
                basicResults.add(r);
            }
        }
        basicResults.sort((a, b) -> {
            if (a.date == null || b.date == null) return 0;
            return b.date.compareTo(a.date);
        });

        List<StopSearchRecord> optimizedResults = optimized.getEthnicitySortedCache().getOrDefault("white", new ArrayList<>());
    }
}
```

Class StopSearchRecord:

```
package main_db;
import java.time.LocalDateTime;
public class StopSearchRecord {
    String type;
    LocalDateTime date;
    String gender;
    String ageRange;
    String selfDefinedEthnicity;
    String officerDefinedEthnicity;
    String legislation;
    String objectOfSearch;
    String outcome;
    Boolean outcomeLinked;
    Boolean removalOfClothing;
    Double latitude;
    Double longitude;
    public StopSearchRecord(String type, LocalDateTime date, String gender, String ageRange,
                            String selfDefinedEthnicity, String officerDefinedEthnicity,
                            String legislation, String objectOfSearch, String outcome,
```

```
                    Boolean outcomeLinked, Boolean removalOfClothing,
                    Double latitude, Double longitude) {
    this.type = type;
    this.date = date;
    this.gender = gender;
    this.ageRange = ageRange;
    this.selfDefinedEthnicity = selfDefinedEthnicity;
    this.officerDefinedEthnicity = officerDefinedEthnicity;
    this.legislation = legislation;
    this.objectOfSearch = objectOfSearch;
    this.outcome = outcome;
    this.outcomeLinked = outcomeLinked;
    this.removalOfClothing = removalOfClothing;
    this.latitude = latitude;
    this.longitude = longitude;
  }
  public String toString() {
    return String.format("[%s] %s, %s, %s, %s, Legislation: %s, Outcome: %s",
        date, gender, ageRange, objectOfSearch, selfDefinedEthnicity, legislation, outcome);
  }
}
```

Class DataLoader:

```
package main_db;
import java.io.*;
import java.time.LocalDateTime;
import java.util.*;
public class DataLoader {
  public List<StopSearchRecord> loadFiles(String[] files) {
    List<StopSearchRecord> records = new ArrayList<>();
    int totalRecords = 0, failedRecords = 0;
    System.out.println("Loading Stop and Search Data...");
    for (String filename : files) {
      File file = new File(filename);
      if (!file.exists()) {
        System.err.println("File not found: " + filename);
        continue;
      }
      try (BufferedReader br = new BufferedReader(new FileReader(file))) {
        String line;
        boolean first = true;
        while ((line = br.readLine()) != null) {
          if (first) { first = false; continue; }
          try {
            StopSearchRecord record = parseCSVLine(line);
            if (record != null) {
              records.add(record);
              totalRecords++;
            }
          } catch (Exception e) {
            failedRecords++;
          }
        }
      } catch (Exception ignore) {}
    }
    System.out.println("Loaded: " + totalRecords);
    System.out.println("Failed: " + failedRecords);
    System.out.println("Total in memory: " + records.size());
    return records;
  }
  private StopSearchRecord parseCSVLine(String line) {
    String[] fields = line.split(",(?=(?:[^\"]*\"[^\"]*\")*[^\"]*$)", -1);
    if (fields.length < 15) return null;
    try {
      StopSearchRecord r = new StopSearchRecord(line, null, line, line, line, line, line, line, null, null, null, null);
      r.type = getField(fields, 0);
      r.date = parseDate(getField(fields, 1));
```

```
        r.gender = getField(fields, 6);
        r.ageRange = getField(fields, 7);
        r.selfDefinedEthnicity = getField(fields, 8);
        r.officerDefinedEthnicity = getField(fields, 9);
        r.legislation = getField(fields, 10);
        r.objectOfSearch = getField(fields, 11);
        r.outcome = getField(fields, 12);
        r.outcomeLinked = parseBoolean(getField(fields, 13));
        r.removalOfClothing = parseBoolean(getField(fields, 14));
        String lat = getField(fields, 4);
        String lon = getField(fields, 5);
        if (!lat.isEmpty() && !lon.isEmpty()) {
            try {
                r.latitude = Double.parseDouble(lat);
                r.longitude = Double.parseDouble(lon);
            } catch (Exception ignored) {}
        }
        return r;
    } catch (Exception e) {
        return null;
    }
  }
  private String getField(String[] f, int i) {
    if (i < f.length && !f[i].isEmpty()) return f[i].trim();
    return "";
  }
  private LocalDateTime parseDate(String d) {
    if (d.isEmpty()) return null;
    try { return LocalDateTime.parse(d.replace("+00:00", "")); }
    catch (Exception e) { return null; }
  }
  private Boolean parseBoolean(String b) {
    if (b.isEmpty()) return null;
    return b.equalsIgnoreCase("True");
  }
}
```

Class FeatureProcessor:

```
package main_db;
import java.util.*;
public class FeatureProcessor {
  List<StopSearchRecord> records;
  private Scanner scanner;
  public FeatureProcessor(List<StopSearchRecord> records, Scanner scanner) {
    this.records = records;
    this.scanner = scanner;
  }
  /* ---------------------- FEATURE A ---------------------- */
  public void featureA() {
    System.out.println("\n--- Distinct Search Purposes ---");
    Set<String> set = new TreeSet<>();
    for (StopSearchRecord r : records)
      if (!r.objectOfSearch.isEmpty())
        set.add(r.objectOfSearch);
    set.forEach(System.out::println);
  }
  /* ---------------------- FEATURE B ---------------------- */
  public void featureB() {
    System.out.println("\n--- Search by Purpose ---");
    Set<String> purposes = new TreeSet<>();
    for (StopSearchRecord r : records)
      if (!r.objectOfSearch.isEmpty())
        purposes.add(r.objectOfSearch);
    List<String> list = new ArrayList<>(purposes);
    for (int i = 0; i < list.size(); i++)
      System.out.println((i + 1) + ". " + list.get(i));
    System.out.print("Choose: ");
```

```
    String input = scanner.nextLine();
    String purpose;
    try {
        purpose = list.get(Integer.parseInt(input) - 1);
    } catch (Exception e) {
        purpose = input;
    }
    List<StopSearchRecord> matches = new ArrayList<>();
    for (StopSearchRecord r : records)
        if (r.objectOfSearch.equalsIgnoreCase(purpose))
            matches.add(r);
    System.out.println("Found: " + matches.size());
    System.out.print("Show? (y/n): ");
    if (scanner.nextLine().equalsIgnoreCase("y"))
        matches.forEach(this::printRecord);
}
/* ---------- FEATURE C ---------- */
public void featureC() {
    System.out.println("\n--- Outcome Analysis ---");
    int success = 0, partial = 0, fail = 0;
    List<StopSearchRecord> s1 = new ArrayList<>(), s2 = new ArrayList<>(), s3 = new ArrayList<>();
    for (StopSearchRecord r : records) {
        if (isSuccessful(r)) { success++; s1.add(r); }
        else if (isPartlySuccessful(r)) { partial++; s2.add(r); }
        else if (isUnsuccessful(r)) { fail++; s3.add(r); }
    }
    System.out.println("Successful: " + success);
    System.out.println("Partly: " + partial);
    System.out.println("Unsuccessful: " + fail);
    System.out.print("Show which? (1/2/3/0): ");
    switch (scanner.nextLine()) {
        case "1": s1.forEach(this::printRecord); break;
        case "2": s2.forEach(this::printRecord); break;
        case "3": s3.forEach(this::printRecord); break;
    }
}
private boolean isSuccessful(StopSearchRecord r) {
    String o = r.outcome.toLowerCase();
    boolean positive = o.contains("arrest") || o.contains("caution") || o.contains("penalty") || o.contains("warning");
    return positive && r.outcomeLinked != null && r.outcomeLinked;
}
private boolean isPartlySuccessful(StopSearchRecord r) {
    String o = r.outcome.toLowerCase();
    boolean positive = o.contains("arrest") || o.contains("caution") || o.contains("penalty") || o.contains("warning");
    return positive && (r.outcomeLinked == null || !r.outcomeLinked);
}
private boolean isUnsuccessful(StopSearchRecord r) {
    String o = r.outcome.toLowerCase();
    return o.contains("no further action") || o.isEmpty();
}
/* ---------- FEATURE D ---------- */
public void featureD() {
    System.out.print("Enter month (6-9): ");
    int month = Integer.parseInt(scanner.nextLine());
    Map<String, Integer> freq = new HashMap<>();
    for (StopSearchRecord r : records)
        if (r.date != null && r.date.getMonthValue() == month && !r.legislation.isEmpty())
            freq.merge(r.legislation, 1, Integer::sum);
    String top = findMax(freq);
    if (top == null) { System.out.println("No data."); return; }
    System.out.println("Most frequent legislation: " + top + " (" + freq.get(top) + ")");
}
/* ---------- FEATURE E ---------- */
public void featureE() {
    System.out.println("1. Month-wise");
    System.out.println("2. Legislation-wise");
    System.out.print("Choose: ");
    switch (scanner.nextLine()) {
        case "1": ethnicByMonth(); break;
        case "2": ethnicByLaw(); break;
```

```
        default: System.out.println("Invalid");
    }
}
private void ethnicByMonth() {
    System.out.print("Enter month: ");
    int m = Integer.parseInt(scanner.nextLine());
    Map<String, Integer> map = new HashMap<>();
    for (StopSearchRecord r : records)
        if (r.date != null && r.date.getMonthValue() == m && !r.selfDefinedEthnicity.isEmpty())
            map.merge(r.selfDefinedEthnicity, 1, Integer::sum);
    String group = findMax(map);
    if (group == null) { System.out.println("No data."); return; }
    System.out.println("Highest: " + group + " (" + map.get(group) + ")");
}
private void ethnicByLaw() {
    Set<String> laws = new TreeSet<>();
    for (StopSearchRecord r : records)
        if (!r.legislation.isEmpty()) laws.add(r.legislation);
    List<String> list = new ArrayList<>(laws);
    for (int i = 0; i < list.size(); i++)
        System.out.println((i + 1) + ". " + list.get(i));
    System.out.print("Choose: ");
    String input = scanner.nextLine();
    String law;
    try { law = list.get(Integer.parseInt(input) - 1); }
    catch (Exception e) { law = input; }
    Map<String, Integer> map = new HashMap<>();
    for (StopSearchRecord r : records)
        if (r.legislation.equalsIgnoreCase(law) && !r.selfDefinedEthnicity.isEmpty())
            map.merge(r.selfDefinedEthnicity, 1, Integer::sum);
    String group = findMax(map);
    if (group == null) { System.out.println("No data."); return; }
    System.out.println("Highest: " + group + " (" + map.get(group) + ")");
}
/* ---------------------- FEATURE F ---------------------- */
public void featureF() {
    System.out.println("\n--- Reverse Chronological by Ethnicity ---");
    Set<String> set = new TreeSet<>();
    for (StopSearchRecord r : records)
        if (!r.selfDefinedEthnicity.isEmpty())
            set.add(r.selfDefinedEthnicity);
    List<String> list = new ArrayList<>(set);
    for (int i = 0; i < list.size(); i++)
        System.out.println((i + 1) + ". " + list.get(i));
    System.out.print("Choose: ");
    String input = scanner.nextLine();
    String eth;
    try { eth = list.get(Integer.parseInt(input) - 1); }
    catch (Exception e) { eth = input; }
    List<StopSearchRecord> match = new ArrayList<>();
    for (StopSearchRecord r : records)
        if (r.selfDefinedEthnicity.equalsIgnoreCase(eth) ||
            r.selfDefinedEthnicity.toLowerCase().contains(eth.toLowerCase()))
            match.add(r);
    match.sort((a, b) -> {
        if (a.date == null) return 1;
        if (b.date == null) return -1;
        return b.date.compareTo(a.date);
    });
    match.forEach(this::printRecord);
}
/* ---------------------- FEATURE G ---------------------- */
public void featureG() {
    System.out.println("\n--- Custom Multi-Attribute Search ---");
    System.out.print("Gender: "); String g = scanner.nextLine();
    System.out.print("Age range: "); String a = scanner.nextLine();
    System.out.print("Ethnicity: "); String e = scanner.nextLine();
    System.out.print("Outcome: "); String o = scanner.nextLine();
    System.out.print("Object: "); String obj = scanner.nextLine();
    List<StopSearchRecord> results = new ArrayList<>();
```

```
    for (StopSearchRecord r : records) {
        boolean ok = true;

        String gender = r.gender.trim().toLowerCase();
        String ageRange = r.ageRange.trim().toLowerCase();
        String ethnicity = r.selfDefinedEthnicity.trim().toLowerCase();
        String outcome = r.outcome.trim().toLowerCase();
        String object = r.objectOfSearch.trim().toLowerCase();
        String g_ = g.trim().toLowerCase();
        String a_ = a.trim().toLowerCase();
        String e_ = e.trim().toLowerCase();
        String o_ = o.trim().toLowerCase();
        String obj_ = obj.trim().toLowerCase();
        if (!g_.isEmpty() && !gender.contains(g_)) ok = false;
        if (!a_.isEmpty() && !ageRange.contains(a_)) ok = false;
        if (!e_.isEmpty() && !ethnicity.contains(e_)) ok = false;
        if (!o_.isEmpty() && !outcome.contains(o_)) ok = false;
        if (!obj_.isEmpty() && !object.contains(obj_)) ok = false;
        if (ok) results.add(r);
    }
    System.out.println("Matches: " + results.size());
    System.out.print("Show? (y/n): ");
    if (scanner.nextLine().equalsIgnoreCase("y"))
        results.forEach(this::printRecord);
  }



  /* ---------------------- PROFILING VERSION (NO USER INPUT) ---------------------- */
  public void featureC_profiling() {
    int success = 0, partial = 0, fail = 0;

    for (StopSearchRecord r : records) {
        if (isSuccessful(r)) success++;
        else if (isPartlySuccessful(r)) partial++;
        else if (isUnsuccessful(r)) fail++;
    }
  }
  public void featureB_profiling(String purpose) {
    int count = 0;
    for (StopSearchRecord r : records) {
        if (r.objectOfSearch.equalsIgnoreCase(purpose)) {
            count++;
        }
    }

  }

  /* ---------------------- UTILS ---------------------- */
  private String findMax(Map<String, Integer> map) {
    return map.entrySet().stream()
            .max(Map.Entry.comparingByValue())
            .map(Map.Entry::getKey).orElse(null);
  }
  private void printRecord(StopSearchRecord r) {
    System.out.println("\nDate: " + (r.date != null ? r.date : "Unknown"));
    System.out.println("Gender: " + r.gender);
    System.out.println("Age: " + r.ageRange);
    System.out.println("Ethnicity: " + r.selfDefinedEthnicity);
    System.out.println("Object: " + r.objectOfSearch);
    System.out.println("Legislation: " + r.legislation);
    System.out.println("Outcome: " + r.outcome);
    System.out.println("Type: " + r.type);
  }
}
```

Class MenuHandler:

```
package main_db;
import java.util.*;
public class MenuHandler {
  private Scanner scanner;
  private FeatureProcessor basicProcessor;
  private OptimizedFeatureProcessor optimizedProcessor;
  private List<StopSearchRecord> records;
  public MenuHandler(List<StopSearchRecord> records, Scanner scanner) {
     this.scanner = scanner;
     this.records = records;
     this.basicProcessor = new FeatureProcessor(records, scanner);
  }
  public void showMenu() {
     while (true) {
        System.out.println("\n=== Police Stop & Search Analysis ===");
        System.out.println("1. Basic Version (Linear Search - O(n))");
        System.out.println("2. Optimized Version (Indexed - O(1))");
        System.out.println("X. Exit");
        System.out.print("Choose implementation: ");
        String choice = scanner.nextLine().toUpperCase();
        switch (choice) {
           case "1":
              showBasicMenu();
              break;
           case "2":
              showOptimizedMenu();
              break;
           case "X":
              System.out.println("Goodbye!");
              return;
           default:
              System.out.println("Invalid choice! Please enter 1, 2, or X.");
        }
     }
  }
  private void showBasicMenu() {
     while (true) {
        System.out.println("\n=== BASIC VERSION (Linear Search) ===");
        System.out.println("A. List distinct search purposes");
        System.out.println("B. Search by purpose");
        System.out.println("C. Analyze outcomes");
        System.out.println("D. Most frequent legislation");
        System.out.println("E. Ethnic analysis");
        System.out.println("F. Reverse chronological listing");
        System.out.println("G. Multi-attribute search");
        System.out.println("BACK. Return to main menu");
        System.out.print("Choose feature: ");
        String choice = scanner.nextLine().toUpperCase();
        switch (choice) {
           case "A":
              basicProcessor.featureA();
              break;
           case "B":
              basicProcessor.featureB();
              break;
           case "C":
              basicProcessor.featureC();
              break;
           case "D":
              basicProcessor.featureD();
              break;
           case "E":
              basicProcessor.featureE();
              break;
           case "F":
              basicProcessor.featureF();
              break;
```

```
            case "G":
                basicProcessor.featureG();
                break;
            case "BACK":
                return;
            default:
                System.out.println("Invalid choice! Please enter A-G or BACK.");
        }
    }
}
private void showOptimizedMenu() {
    if (optimizedProcessor == null) {
        System.out.println("Building indexes and caches...");
        optimizedProcessor = new OptimizedFeatureProcessor(records, scanner);
    }
    while (true) {
        System.out.println("\n=== OPTIMIZED VERSION (Indexed & Cached) ===");
        System.out.println("A. List distinct search purposes (O(1))");
        System.out.println("B. Search by purpose (O(1) indexed)");
        System.out.println("C. Analyze outcomes (Indexed filtering)");
        System.out.println("D. Most frequent legislation (Cached results)");
        System.out.println("E. Ethnic analysis (Cached/Indexed)");
        System.out.println("F. Reverse chronological listing (Pre-sorted)");
        System.out.println("G. Multi-attribute search (Optimized filtering)");
        System.out.println("BACK. Return to main menu");
        System.out.print("Choose feature: ");
        String choice = scanner.nextLine().toUpperCase();
        switch (choice) {
            case "A":
                optimizedProcessor.featureA();
                break;
            case "B":
                optimizedProcessor.featureB();
                break;
            case "C":
                optimizedProcessor.featureC();
                break;
            case "D":
                optimizedProcessor.featureD();
                break;
            case "E":
                optimizedProcessor.featureE();
                break;
            case "F":
                optimizedProcessor.featureF();
                break;
            case "G":
                optimizedProcessor.featureG();
                break;
            case "EXIT":
                return;
            default:
                System.out.println("Invalid choice! Please enter A-G or BACK.");
        }
    }
}
}
```

Class OptimizedFeatureProcessor:

```
package main_db;
import java.util.*;
import java.util.stream.Collectors;
public class OptimizedFeatureProcessor {
  private List<StopSearchRecord> records;
  private Scanner scanner;
```

```
private Map<String, List<StopSearchRecord>> purposeIndex;
private Map<String, List<StopSearchRecord>> ethnicityIndex;
private Map<String, List<StopSearchRecord>> legislationIndex;
private Map<String, List<StopSearchRecord>> genderIndex;
private Map<String, List<StopSearchRecord>> ageRangeIndex;
private Map<String, List<StopSearchRecord>> outcomeIndex;

// Caches for frequent queries
private Map<String, Map<String, Integer>> monthlyLegislationCache;
private Map<String, Map<String, Integer>> monthlyEthnicityCache;
private Map<String, String> mostFrequentLegislationCache;

private List<StopSearchRecord> recordsSortedByDate;
private Map<String, List<StopSearchRecord>> ethnicitySortedCache;
public OptimizedFeatureProcessor(List<StopSearchRecord> records, Scanner scanner) {
    this.records = records;
    this.scanner = scanner;
    buildIndexes();
    buildCaches();
    preSortData();
}
/* ========== INDEX BUILDING ========== */
private void buildIndexes() {
    System.out.println("Building indexes...");

    purposeIndex = new HashMap<>();
    ethnicityIndex = new HashMap<>();
    legislationIndex = new HashMap<>();
    genderIndex = new HashMap<>();
    ageRangeIndex = new HashMap<>();
    outcomeIndex = new HashMap<>();
    for (StopSearchRecord record : records) {
        // Index by purpose (O(1) lookup)
        if (!record.objectOfSearch.isEmpty()) {
            purposeIndex.computeIfAbsent(record.objectOfSearch.toLowerCase(),
                k -> new ArrayList<>()).add(record);
        }

        // Index by ethnicity (O(1) lookup)
        if (!record.selfDefinedEthnicity.isEmpty()) {
            ethnicityIndex.computeIfAbsent(record.selfDefinedEthnicity.toLowerCase(),
                k -> new ArrayList<>()).add(record);
        }

        // Index by legislation (O(1) lookup)
        if (!record.legislation.isEmpty()) {
            legislationIndex.computeIfAbsent(record.legislation.toLowerCase(),
                k -> new ArrayList<>()).add(record);
        }

        // Index by gender (O(1) lookup)
        if (!record.gender.isEmpty()) {
            genderIndex.computeIfAbsent(record.gender.toLowerCase(),
                k -> new ArrayList<>()).add(record);
        }

        // Index by age range (O(1) lookup)
        if (!record.ageRange.isEmpty()) {
            ageRangeIndex.computeIfAbsent(record.ageRange.toLowerCase(),
                k -> new ArrayList<>()).add(record);
        }

        // Index by outcome (O(1) lookup)
        if (!record.outcome.isEmpty()) {
            outcomeIndex.computeIfAbsent(record.outcome.toLowerCase(),
                k -> new ArrayList<>()).add(record);
        }
    }
```

```
}
/* ========== CACHE BUILDING ========== */
private void buildCaches() {
    System.out.println("Building caches...");

    monthlyLegislationCache = new HashMap<>();
    monthlyEthnicityCache = new HashMap<>();
    mostFrequentLegislationCache = new HashMap<>();
    for (StopSearchRecord record : records) {
        if (record.date != null) {
            String monthKey = record.date.getMonthValue() + "-" + record.date.getYear();

            if (!record.legislation.isEmpty()) {
                monthlyLegislationCache
                    .computeIfAbsent(monthKey, k -> new HashMap<>())
                    .merge(record.legislation, 1, Integer::sum);
            }

            if (!record.selfDefinedEthnicity.isEmpty()) {
                monthlyEthnicityCache
                    .computeIfAbsent(monthKey, k -> new HashMap<>())
                    .merge(record.selfDefinedEthnicity, 1, Integer::sum);
            }
        }
    }

    for (String monthKey : monthlyLegislationCache.keySet()) {
        mostFrequentLegislationCache.put(monthKey,
            findMaxKey(monthlyLegislationCache.get(monthKey)));
    }
}
/* ========== PRE-SORTING ========== */
private void preSortData() {
    System.out.println("Pre-sorting data...");


    recordsSortedByDate = new ArrayList<>(records);
    recordsSortedByDate.sort((a, b) -> {
        if (a.date == null) return 1;
        if (b.date == null) return -1;
        return b.date.compareTo(a.date);
    });


    ethnicitySortedCache = new HashMap<>();
    for (Map.Entry<String, List<StopSearchRecord>> entry : ethnicityIndex.entrySet()) {
        List<StopSearchRecord> sorted = new ArrayList<>(entry.getValue());
        sorted.sort((a, b) -> {
            if (a.date == null) return 1;
            if (b.date == null) return -1;
            return b.date.compareTo(a.date);
        });
        ethnicitySortedCache.put(entry.getKey(), sorted);
    }
}
/* ========== OPTIMIZED FEATURE A (O(1)) ========== */
public void featureA() {
    System.out.println("\n--- Distinct Search Purposes (Optimized) ---");
    System.out.println("Found " + purposeIndex.size() + " distinct purposes:");
    purposeIndex.keySet().forEach(purpose ->
        System.out.println(" - " + purpose + " (" + purposeIndex.get(purpose).size() + " records)"));
}
/* ========== OPTIMIZED FEATURE B (O(1)) ========== */
public void featureB() {
    System.out.println("\n--- Search by Purpose (Optimized) ---");


    List<String> purposes = new ArrayList<>(purposeIndex.keySet());
    for (int i = 0; i < purposes.size(); i++) {
        System.out.println((i + 1) + ". " + purposes.get(i) +
```

```
            " (" + purposeIndex.get(purposes.get(i)).size() + " records)");
    }
    System.out.print("Choose purpose number or name: ");
    String input = scanner.nextLine();
    String purpose;
    try {
        purpose = purposes.get(Integer.parseInt(input) - 1);
    } catch (Exception e) {
        purpose = input.toLowerCase();
    }

    List<StopSearchRecord> matches = purposeIndex.getOrDefault(purpose, new ArrayList<>());

    System.out.println("Found: " + matches.size() + " records (O(1) lookup)");
    System.out.print("Show details? (y/n): ");
    if (scanner.nextLine().equalsIgnoreCase("y")) {
        matches.forEach(this::printRecord);
    }
}
/* ========== OPTIMIZED FEATURE C (O(1) outcome lookup) ========== */
public void featureC() {
    System.out.println("\n--- Outcome Analysis (Optimized) ---");

    List<StopSearchRecord> successful = new ArrayList<>();
    List<StopSearchRecord> partlySuccessful = new ArrayList<>();
    List<StopSearchRecord> unsuccessful = new ArrayList<>();

    for (List<StopSearchRecord> outcomeRecords : outcomeIndex.values()) {
        for (StopSearchRecord record : outcomeRecords) {
            if (isSuccessful(record)) successful.add(record);
            else if (isPartlySuccessful(record)) partlySuccessful.add(record);
            else if (isUnsuccessful(record)) unsuccessful.add(record);
        }
    }
    System.out.println("Successful: " + successful.size() + " (using outcome index)");
    System.out.println("Partly: " + partlySuccessful.size());
    System.out.println("Unsuccessful: " + unsuccessful.size());
    System.out.print("Show which? (1/2/3/0): ");
    switch (scanner.nextLine()) {
        case "1": successful.forEach(this::printRecord); break;
        case "2": partlySuccessful.forEach(this::printRecord); break;
        case "3": unsuccessful.forEach(this::printRecord); break;
    }

}
/* ========== OPTIMIZED FEATURE D (O(1) cache lookup) ========== */
public void featureD() {
    System.out.print("Enter month (6-9): ");
    int month = Integer.parseInt(scanner.nextLine());

    System.out.print("Enter year (e.g., 2025): ");
    int year = Integer.parseInt(scanner.nextLine());

    String monthKey = month + "-" + year;
    Map<String, Integer> legislationCounts = monthlyLegislationCache.get(monthKey);

    if (legislationCounts == null) {
        System.out.println("No data for " + monthKey);
        return;
    }
    String topLegislation = mostFrequentLegislationCache.get(monthKey);
    System.out.println("Most frequent legislation: " + topLegislation +
        " (" + legislationCounts.get(topLegislation) + " records) - (Cached result)");

    Map<String, Integer> successfulCounts = new HashMap<>();
    for (StopSearchRecord record : records) {
        if (record.date != null &&
            record.date.getMonthValue() == month &&
            record.date.getYear() == year &&
            !record.legislation.isEmpty() &&
```

```
            isSuccessful(record)) {
            successfulCounts.merge(record.legislation, 1, Integer::sum);
        }
    }
    String topSuccessful = findMaxKey(successfulCounts);
    if (topSuccessful != null) {
        System.out.println("Most successful legislation: " + topSuccessful +
            " (" + successfulCounts.get(topSuccessful) + " successful records)");
    }
}
/* ========== OPTIMIZED FEATURE E (O(1) cache lookup) ========== */
public void featureE() {
    System.out.println("1. Month-wise (Cached)");
    System.out.println("2. Legislation-wise (Indexed)");
    System.out.print("Choose: ");
    switch (scanner.nextLine()) {
        case "1": ethnicByMonthOptimized(); break;
        case "2": ethnicByLawOptimized(); break;
        default: System.out.println("Invalid");
    }
}
private void ethnicByMonthOptimized() {
    System.out.print("Enter month: ");
    int month = Integer.parseInt(scanner.nextLine());
    System.out.print("Enter year: ");
    int year = Integer.parseInt(scanner.nextLine());

    String monthKey = month + "-" + year;
    Map<String, Integer> ethnicityCounts = monthlyEthnicityCache.get(monthKey);

    if (ethnicityCounts == null) {
        System.out.println("No data for " + monthKey);
        return;
    }
    String topEthnicity = findMaxKey(ethnicityCounts);
    System.out.println("Highest ethnicity group: " + topEthnicity +
        " (" + ethnicityCounts.get(topEthnicity) + " records) - (Cached result)");
    System.out.print("Show details? (y/n): ");
    if (scanner.nextLine().equalsIgnoreCase("y")) {
        List<StopSearchRecord> ethnicRecords = ethnicityIndex.getOrDefault(
            topEthnicity.toLowerCase(), new ArrayList<>());
        ethnicRecords.forEach(this::printRecord);
    }
}
private void ethnicByLawOptimized() {
    List<String> laws = new ArrayList<>(legislationIndex.keySet());
    for (int i = 0; i < laws.size(); i++) {
        System.out.println((i + 1) + ". " + laws.get(i) +
            " (" + legislationIndex.get(laws.get(i)).size() + " records)");
    }
    System.out.print("Choose legislation: ");
    String input = scanner.nextLine();
    String law;
    try {
        law = laws.get(Integer.parseInt(input) - 1);
    } catch (Exception e) {
        law = input.toLowerCase();
    }

    List<StopSearchRecord> lawRecords = legislationIndex.getOrDefault(law, new ArrayList<>());
    Map<String, Integer> ethnicityCounts = new HashMap<>();
    for (StopSearchRecord record : lawRecords) {
        if (!record.selfDefinedEthnicity.isEmpty()) {
            ethnicityCounts.merge(record.selfDefinedEthnicity, 1, Integer::sum);
        }
    }
    String topEthnicity = findMaxKey(ethnicityCounts);
    if (topEthnicity != null) {
        System.out.println("Highest ethnicity for " + law + ": " + topEthnicity +
            " (" + ethnicityCounts.get(topEthnicity) + " records) - (Indexed lookup)");
```

```
    }
}
/* ========== OPTIMIZED FEATURE F (O(1) pre-sorted) ========== */
public void featureF() {
    System.out.println("\n--- Reverse Chronological by Ethnicity (Optimized) ---");
    List<String> ethnicities = new ArrayList<>(ethnicityIndex.keySet());
    for (int i = 0; i < ethnicities.size(); i++) {
        System.out.println((i + 1) + ". " + ethnicities.get(i) +
            " (" + ethnicityIndex.get(ethnicities.get(i)).size() + " records)");
    }
    System.out.print("Choose ethnicity: ");
    String input = scanner.nextLine();
    String ethnicity;
    try {
        ethnicity = ethnicities.get(Integer.parseInt(input) - 1);
    } catch (Exception e) {
        ethnicity = input.toLowerCase();
    }
    List<StopSearchRecord> matches = ethnicitySortedCache.getOrDefault(ethnicity, new ArrayList<>());

    System.out.println("Found: " + matches.size() + " records (Pre-sorted, O(1) access)");
    matches.forEach(this::printRecord);
}
/* ========== OPTIMIZED FEATURE G (O(min(s1,s2,...)) ========== */
public void featureG() {
    System.out.println("\n--- Custom Multi-Attribute Search (Optimized) ---");
    System.out.print("Gender: "); String gender = scanner.nextLine();
    System.out.print("Age range: "); String ageRange = scanner.nextLine();
    System.out.print("Ethnicity: "); String ethnicity = scanner.nextLine();
    System.out.print("Outcome: "); String outcome = scanner.nextLine();
    System.out.print("Object: "); String object = scanner.nextLine();

    List<StopSearchRecord> results = null;
    String smallestIndex = findSmallestIndex(gender, ageRange, ethnicity, outcome, object);

    switch (smallestIndex) {
        case "gender":
            results = genderIndex.getOrDefault(gender.toLowerCase(), new ArrayList<>());
            break;
        case "ageRange":
            results = ageRangeIndex.getOrDefault(ageRange.toLowerCase(), new ArrayList<>());
            break;
        case "ethnicity":
            results = ethnicityIndex.getOrDefault(ethnicity.toLowerCase(), new ArrayList<>());
            break;
        case "outcome":
            results = outcomeIndex.getOrDefault(outcome.toLowerCase(), new ArrayList<>());
            break;
        case "object":
            results = purposeIndex.getOrDefault(object.toLowerCase(), new ArrayList<>());
            break;
        default:
            results = new ArrayList<>(records);
    }

    List<StopSearchRecord> finalResults = results.stream()
        .filter(r -> gender.isEmpty() || r.gender.equalsIgnoreCase(gender))
        .filter(r -> ageRange.isEmpty() || r.ageRange.equalsIgnoreCase(ageRange))
        .filter(r -> ethnicity.isEmpty() || r.selfDefinedEthnicity.toLowerCase().contains(ethnicity.toLowerCase()))
        .filter(r -> outcome.isEmpty() || r.outcome.toLowerCase().contains(outcome.toLowerCase()))
        .filter(r -> object.isEmpty() || r.objectOfSearch.toLowerCase().contains(object.toLowerCase()))
        .collect(Collectors.toList());
    System.out.println("Matches: " + finalResults.size() + " (Optimized filtering)");
    System.out.print("Show? (y/n): ");
    if (scanner.nextLine().equalsIgnoreCase("y")) {
        finalResults.forEach(this::printRecord);
    }
}
private String findSmallestIndex(String gender, String ageRange, String ethnicity, String outcome, String object) {
    Map<String, Integer> sizes = new HashMap<>();
```

```
    if (!gender.isEmpty()) sizes.put("gender", genderIndex.getOrDefault(gender.toLowerCase(), new ArrayList<>()).size());
    if (!ageRange.isEmpty()) sizes.put("ageRange", ageRangeIndex.getOrDefault(ageRange.toLowerCase(), new ArrayList<>()).size());
    if (!ethnicity.isEmpty()) sizes.put("ethnicity", ethnicityIndex.getOrDefault(ethnicity.toLowerCase(), new ArrayList<>()).size());
    if (!outcome.isEmpty()) sizes.put("outcome", outcomeIndex.getOrDefault(outcome.toLowerCase(), new ArrayList<>()).size());
    if (!object.isEmpty()) sizes.put("object", purposeIndex.getOrDefault(object.toLowerCase(), new ArrayList<>()).size());

    return sizes.entrySet().stream()
        .min(Map.Entry.comparingByValue())
        .map(Map.Entry::getKey)
        .orElse("all");
  }
  /* ========== HELPER METHODS ========== */
  private boolean isSuccessful(StopSearchRecord r) {
    String o = r.outcome.toLowerCase();
    boolean positive = o.contains("arrest") || o.contains("caution") || o.contains("penalty") || o.contains("warning");
    return positive && r.outcomeLinked != null && r.outcomeLinked;
  }
  private boolean isPartlySuccessful(StopSearchRecord r) {
    String o = r.outcome.toLowerCase();
    boolean positive = o.contains("arrest") || o.contains("caution") || o.contains("penalty") || o.contains("warning");
    return positive && (r.outcomeLinked == null || !r.outcomeLinked);
  }
  private boolean isUnsuccessful(StopSearchRecord r) {
    String o = r.outcome.toLowerCase();
    return o.contains("no further action") || o.isEmpty();
  }
  private String findMaxKey(Map<String, Integer> map) {
    return map.entrySet().stream()
          .max(Map.Entry.comparingByValue())
          .map(Map.Entry::getKey)
          .orElse(null);
  }

  /* ========== GETTER METHODS FOR PROFILING ========== */
public Map<String, List<StopSearchRecord>> getPurposeIndex() {
  return purposeIndex;
}
public Map<String, List<StopSearchRecord>> getEthnicityIndex() {
  return ethnicityIndex;
}
public Map<String, List<StopSearchRecord>> getLegislationIndex() {
  return legislationIndex;
}
public Map<String, List<StopSearchRecord>> getGenderIndex() {
  return genderIndex;
}
public Map<String, List<StopSearchRecord>> getAgeRangeIndex() {
  return ageRangeIndex;
}
public Map<String, List<StopSearchRecord>> getOutcomeIndex() {
  return outcomeIndex;
}
public Map<String, Map<String, Integer>> getMonthlyLegislationCache() {
  return monthlyLegislationCache;
}
public Map<String, Map<String, Integer>> getMonthlyEthnicityCache() {
  return monthlyEthnicityCache;
}
public Map<String, String> getMostFrequentLegislationCache() {
  return mostFrequentLegislationCache;
}
public List<StopSearchRecord> getRecordsSortedByDate() {
  return recordsSortedByDate;
}
public Map<String, List<StopSearchRecord>> getEthnicitySortedCache() {
  return ethnicitySortedCache;
}
  private void printRecord(StopSearchRecord r) {
    System.out.println("\nDate: " + (r.date != null ? r.date : "Unknown"));
```

```
    System.out.println("Gender: " + r.gender);
    System.out.println("Age: " + r.ageRange);
    System.out.println("Ethnicity: " + r.selfDefinedEthnicity);
    System.out.println("Object: " + r.objectOfSearch);
    System.out.println("Legislation: " + r.legislation);
    System.out.println("Outcome: " + r.outcome);
    System.out.println("Type: " + r.type);
    System.out.println("---");
  }
}
```

9 Testing

```
Loading Stop and Search Data...
Loaded: 24092
Failed: 0
Total in memory: 24092

=== Police Stop & Search Analysis ===
1. Basic Version (Linear Search - O(n))
2. Optimized Version (Indexed - O(1))
X. Exit
Choose implementation: 1

=== BASIC VERSION (Linear Search) ===
A. List distinct search purposes
B. Search by purpose
C. Analyze outcomes
D. Most frequent legislation
E. Ethnic analysis
F. Reverse chronological listing
G. Multi-attribute search
BACK. Return to main menu
Choose feature:
```

```
=== BASIC VERSION (Linear Search) ===
A. List distinct search purposes
B. Search by purpose
C. Analyze outcomes
D. Most frequent legislation
E. Ethnic analysis
F. Reverse chronological listing
G. Multi-attribute search
BACK. Return to main menu
Choose feature: b

--- Search by Purpose ---
1. Article for use in theft
2. Articles for use in criminal damage
3. Controlled drugs
4. Evidence of offences under the Act
5. Firearms
6. Fireworks
7. Offensive weapons
8. Psychoactive substances
9. Stolen goods
Choose: 7
Found: 423
Show? (y/n):
```

```
Date: 2025-09-30T19:37
Gender: Male
Age: 25-34
Ethnicity: White - English/Welsh/Scottish/Northern Irish/British
Object: Offensive weapons
Legislation: Police and Criminal Evidence Act 1984 (section 1)
Outcome: A no further action disposal
Type: Person search

Date: 2025-09-30T21:14:28
Gender: Male
Age:
Ethnicity: White - English/Welsh/Scottish/Northern Irish/British
Object: Offensive weapons
Legislation: Police and Criminal Evidence Act 1984 (section 1)
Outcome: Arrest
Type: Person search
```

```
=== BASIC VERSION (Linear Search) ===
A. List distinct search purposes
B. Search by purpose
C. Analyze outcomes
D. Most frequent legislation
E. Ethnic analysis
F. Reverse chronological listing
G. Multi-attribute search
BACK. Return to main menu
Choose feature: g

--- Custom Multi-Attribute Search ---
Gender: Male
Age range: 18-24
Ethnicity: white
Outcome:
Object: controlled drugs
Matches: 236
Show? (y/n):
```

```
=== OPTIMIZED VERSION (Indexed & Cached) ===
A. List distinct search purposes (O(1))
B. Search by purpose (O(1) indexed)
C. Analyze outcomes (Indexed filtering)
D. Most frequent legislation (Cached results)
E. Ethnic analysis (Cached/Indexed)
F. Reverse chronological listing (Pre-sorted)
G. Multi-attribute search (Optimized filtering)
BACK. Return to main menu
Choose feature: c

--- Outcome Analysis (Optimized) ---
Successful: 363 (using outcome index)
Partly: 243
Unsuccessful: 3448
Show which? (1/2/3/0): 2

Date: 2025-06-01T10:00
Gender: Female
Age: 25-34
Ethnicity: White - English/Welsh/Scottish/Northern Irish/British
Object: Controlled drugs
Legislation: Misuse of Drugs Act 1971 (section 23)
Outcome: Arrest
Type: Person and Vehicle search
---

Date: 2025-06-02T14:05
Gender: Male
Age:
Ethnicity: White - English/Welsh/Scottish/Northern Irish/British
Object: Controlled drugs
Legislation: Misuse of Drugs Act 1971 (section 23)
Outcome: Arrest
Type: Person and Vehicle search
```

```
Output - Run (database)
Article for use in theft
Articles for use in criminal damage
Controlled drugs
Evidence of offences under the Act
Firearms
Fireworks
Offensive weapons
Psychoactive substances
Stolen goods

--- Distinct Search Purposes (Optimized) ---
Found 9 distinct purposes:
 - psychoactive substances (2 records)
 - articles for use in criminal damage (187 records)
 - firearms (73 records)
 - offensive weapons (423 records)
 - evidence of offences under the act (595 records)
 - article for use in theft (390 records)
 - fireworks (37 records)
 - stolen goods (225 records)
 - controlled drugs (3196 records)
? Running Feature B (Search by Purpose)...
? Running Feature C (Outcome Analysis)...

--- Outcome Analysis ---
Successful: 363
Partly: 243
Unsuccessful: 22871
```

Feature	Description	Test performed	Expected outcome	Result
A	List distinct search purposes	Checked total purposes from the dataset	All values should be displayed	pass
B	Search by purpose	Searched for “controlled drugs”	All records should be printed	pass
C	Analyze outcomes	Successful/Partial/unsuccessful	Logical grouping by outcome	pass
D	Most frequent legislation	Entered month	Displays most frequent legislation	pass
E	Ethnic analysis	Checked both month and legislation	Highest ethnicity group displayed	pass
F	Reverse chronological listing	Chose “white”	Records sorted by date	pass
G	Multi-attribute search	Entered multiple filters	Correct filtered results	pass

10 Conclusion

In conclusion, this project successfully developed a Java-based system for analyzing Police Stop and Search data using both basic and optimized approaches. It efficiently loads, processes and analyzes datasets through modular classes such as “DataLoader”, “StopSearchRecord” and “MenuHandler”, providing clear information and user interaction. processes, and analyzes datasets through modular classes such as `DataLoader`, `StopSearchRecord`, and `MenuHandler`, providing clear insights and user interaction. This project enhanced our understanding of Java programming, data management, and algorithm optimization while demonstrating the importance of structured and maintainable code design. improved my understanding of Java programming, data handling, and algorithm optimization while demonstrating the importance of structured, maintainable code design.