Code

Rohin Arya, Y11 HL DP Computer Science, May 31

Code also available on GitHub.

App.Java

```
import java.io.File;
import java.io.FileWriter;
import java.util.Base64;
import java.util.Scanner;

public class App {

  // One global system scanner
  private static Scanner s = new Scanner(System.in);

  /**
   * Caesar encrypt
   * @param plain
   * precondition: plain is a string
   * postcondition: returns an encrypted string
   */
  private static String caesarCipherEncrypt(String plain) {
    String b64encoded =
Base64.getEncoder().encodeToString(plain.getBytes());
    String reverse = new StringBuffer(b64encoded).reverse().toString(); //
Reverse the base 64 encoded
    StringBuilder encrypted = new StringBuilder();
    for (int i = 0; i < reverse.length(); i++) {
      encrypted.append((char)(reverse.charAt(i) + 8)); // Add predefined
offset of 8 to each character
    }
    return encrypted.toString();
  }

  // Main entry point
  public static void main(String[] args) throws Exception {
    // Flush terminal
    System.out.print("\033[H\033[2J");
    System.out.flush();

    try {
      // Check if password file exists
      File file = new File("pass.ini");
      if (!file.exists()) {
        System.out.println("🎉 - Welcome to Client Manager 1.0. This seems
to be your first time! Please create a password:");
        String password = s.nextLine(); // Get password
```

```
        FileWriter fw = new FileWriter(file); // Get file writer
        fw.write(caesarCipherEncrypt(password)); // Write encrypted
password to file
        fw.close();
      }
      else {
        Scanner fileScanner = new Scanner(file); // Get file reader
        String password = fileScanner.nextLine(); // Get password
        fileScanner.close(); // Close file reader


        System.out.println("🔑 - Please enter your password:");
        String password2 = s.nextLine(); // Get password

        if (!(password).equals(caesarCipherEncrypt(password2))) {  //
Check if password encryped doesnt match saved
          System.out.println("🗝 - Incorrect password");
          System.exit(0);
        }
      }

      // Start loop class
      Loop loop = new Loop();
      loop.startLoop(s); // Pass scanner to loop

      s.close(); // Close scanner when done loop

    } catch (Exception e) { // Simple error handling
      System.out.println("An exception occured: " + e);
    }
  }
}
```

Client.java

```
import java.io.File;
import java.io.FileWriter;
import java.util.Scanner;

public class Client {

  // Fields
  private String name;
  private String address;
  private String phone;
  private String email;
  private String note;
  private String id;
  private boolean deleted = false;

  /**
```

```
     * Constructor for Client with ID
     * @param id
     */
    public Client(String id) {
      // Get data from file
      File file = new File(id + ".txt");

      if (file.exists()) {
        try {
          Scanner fileScanner = new Scanner(file);
          this.name = fileScanner.nextLine();
          this.address = fileScanner.nextLine();
          this.phone = fileScanner.nextLine();
          this.email = fileScanner.nextLine();
          this.note = fileScanner.nextLine();
          fileScanner.close();
        } catch (Exception e) {
          System.out.println("Error: " + e.getMessage());
        }
      }
      else {
        System.out.println("Client does not exist with this ID. A new empty
client will be created.");
        this.name = "";
        this.address = "";
        this.phone = "";
        this.email = "";
        this.note = "";
      }

      this.id = id;
    }

    /**
     * Second constructor for Client without ID
     */
    public Client() { // Create new client
      this.id = String.valueOf(System.currentTimeMillis());
      this.name = "";
      this.address = "";
      this.phone = "";
      this.email = "";
      this.note = "";
    }

    /**
     * Save client to file
     * precondition: Client has an ID
     * postcondition: Client is saved to file
     */
    public void save() {
      if (this.deleted) {
        throw new RuntimeException("Client is deleted"); // Throw error
      }
```

```
    // Delete the file if it already exists:
    File file = new File(this.id + ".txt");
    if (file.exists()) {
      file.delete();
    }

    // Create a new file of [id].txt
    try {
      file.createNewFile();
    } catch (Exception e) {
      System.out.println("Error: " + e.getMessage());
    }

    // Write the client data to the file
    try {
      FileWriter writer = new FileWriter(file);
      writer.write(this.name + "\n");
      writer.write(this.address + "\n");
      writer.write(this.phone + "\n");
      writer.write(this.email + "\n");
      writer.write(this.note + "\n");
      writer.close();
    } catch (Exception e) {
      System.out.println("Error: " + e.getMessage());
    }

    System.out.println("✅ - Client saved successfully.");
  }

  /**
   * Delete client from file
   * precondition: Client has an ID
   * postcondition: Client is deleted from file
   */
  public void delete() {
    if (this.deleted) {
      throw new RuntimeException("Client is deleted"); // Throw error
    }

    // Get file
    File file = new File(this.id + ".txt");
    // If file exists, delete it
    if (file.exists()) {
      file.delete();
    }
    this.deleted = true;
    System.out.println("✅ - Client deleted successfully.");
  }

  /**
   * Get client deleted status
   */
  public boolean isDeleted() {
```

```
    return deleted;
  }

  // Getters

  /**
   * Get client name
   */
  public String getName() {
    return this.name;
  }

  /**
   * Get client address
   */
  public String getAddress() {
    return address;
  }

  /**
   * Get client phone
   */
  public String getPhone() {
    return phone;
  }

  /**
   * Get client email
   */
  public String getEmail() {
    return email;
  }

  /**
   * Get client note
   */
  public String getNote() {
    return note;
  }

  /**
   * Get client ID
   */
  public String getId() {
    return id;
  }

  // Setters

  /**
   * Set client name
   * @param name
   */
  public void setName(String name) {
```

```
    // Make sure name is not empty
    if (name.isEmpty()) {
      throw new IllegalArgumentException("Name cannot be empty");
    }

    // Make sure name is not longer than 100 characters
    if (name.length() > 100) {
      throw new IllegalArgumentException("Name cannot be longer than 100
characters");
    }

    // Make sure name does not start or end with a space
    if (name.charAt(0) == ' ' || name.charAt(name.length() - 1) == ' ') {
      throw new IllegalArgumentException("Name cannot start or end with a
space");
    }

    // All good
    this.name = name;
  }

  /**
   * Set client address
   * @param address
   */
  public void setAddress(String address) {
    // Make sure address is not empty
    if (address.isEmpty()) {
      throw new IllegalArgumentException("Address cannot be empty");
    }

    // Make sure address is not longer than 200 characters
    if (address.length() > 200) {
      throw new IllegalArgumentException("Address cannot be longer than
200 characters");
    }

    // All good
    this.address = address;
  }

  /**
   * Set client phone
   * @param phone
   */
  public void setPhone(String phone) {
    // Make sure phone is not empty
    if (phone.isEmpty()) {
      throw new IllegalArgumentException("Phone cannot be empty");
    }

    // Make sure phone is not longer than 20 characters
    if (phone.length() > 20) {
      throw new IllegalArgumentException("Phone cannot be longer than 20
```

```
characters");
    }

    // Make sure phone does not start or end with a space
    if (phone.charAt(0) == ' ' || phone.charAt(phone.length() - 1) == ' ')
{
      throw new IllegalArgumentException("Phone cannot start or end with a
space");
    }

    // Make sure phone only contains numbers and spaces and dashes
    for (int i = 0; i < phone.length(); i++) {
      if (!Character.isDigit(phone.charAt(i)) && phone.charAt(i) != ' ' &&
phone.charAt(i) != '-') {
        throw new IllegalArgumentException("Phone can only contain
numbers, spaces, and dashes");
      }
    }

    // All good
    this.phone = phone;
  }

  /**
   * Set client email
   * @param email
   */
  public void setEmail(String email) {
    // Make sure email is not empty
    if (email.isEmpty()) {
      throw new IllegalArgumentException("Email cannot be empty");
    }

    // Make sure email is not longer than 100 characters
    if (email.length() > 100) {
      throw new IllegalArgumentException("Email cannot be longer than 100
characters");
    }

    // Make sure email does not start or end with a space
    if (email.charAt(0) == ' ' || email.charAt(email.length() - 1) == ' ')
{
      throw new IllegalArgumentException("Email cannot start or end with a
space");
    }

    // Make sure email only contains letters, numbers, and spaces, and @
    for (int i = 0; i < email.length(); i++) {
      if (!Character.isLetter(email.charAt(i)) &&
!Character.isDigit(email.charAt(i)) && email.charAt(i) != ' ' &&
email.charAt(i) != '@' && email.charAt(i) != '.') {
        throw new IllegalArgumentException("Email can only contain
letters, numbers, spaces, @, and period.");
      }
```

```
    }

    // All good
    this.email = email;
  }

  /**
   * Set client note
   * @param note
   */
  public void setNote(String note) {
    // Make sure note is not longer than 1000 characters
    if (note.length() > 1000) {
      throw new IllegalArgumentException("Note cannot be longer than 1000
characters");
    }

    // All good
    this.note = note;
  }
}
```

Loop.java

```
import java.util.ArrayList;
import java.util.Scanner;


public class Loop {

  // Fields
  private boolean running = true;
  private boolean listRunning = false;
  private boolean filterRunning = false;
  private boolean clientRunning = false;

  /*
   * Start loop of presenting options to user.
   */
  public void startLoop(Scanner s) {
    clearTerminal(); // Flush terminal

    while (running) { // Loop until user quits
      System.out.println("Available options:");
      System.out.println("1️⃣ - 🟢 - New Client");
      System.out.println("2️⃣ - 📃 - Manage Clients");
      System.out.println("3️⃣ - 🔎 - Search Clients");
      System.out.println("4️⃣ - ⭕ - New Client Loop");
      System.out.println("5️⃣ - ❌ - Exit Program");
      System.out.print("Input option: ");
      int option = s.nextInt();
```

```
        clearTerminal();

        if (option == 1) {
          // Add client
          s.nextLine();
          System.out.print("Name: ");
          String name = s.nextLine(); // Get name
          System.out.print("Address: ");
          String address = s.nextLine(); // Get address
          System.out.print("Phone: ");
          String phone = s.nextLine(); // Get phone
          System.out.print("Email: ");
          String email = s.nextLine(); // Get email
          System.out.print("Note: ");
          String note = s.nextLine(); // Get note

          // Create new client with auto-generated ID
          // Populate with given details
          Client client = new Client();
          client.setName(name);
          client.setAddress(address);
          client.setPhone(phone);
          client.setEmail(email);
          client.setNote(note);
          client.save(); // Save client to file

          clearTerminal();
          System.out.println("✅ - Client added successfully.");
        }
        else if (option == 2) {
          // Show list of all clients
          listRunning = true;
          Search search = new Search();
          while (listRunning) { // Loop until user quits
            clearTerminal();
            search.refreshSearch(); // Make sure data is up to date.
            completeList(search.getClients(), s); // Show list of clients
          }
        }
        else if (option == 3) {
          // Search clients
          Search search = new Search();
          filterRunning = true;
          while (filterRunning) {
            search.refreshSearch();
            System.out.println("Available filters:");
            System.out.println("1️⃣ - Name");
            System.out.println("2️⃣ - Address");
            System.out.println("3️⃣ - Phone");
            System.out.println("4️⃣ - Email");
            System.out.println("5️⃣ - Note");
            System.out.println("6️⃣ - Suspicious");
            System.out.println("Available options:");
            System.out.println("7️⃣ - Print results (" + search.getSize() +
```

```
"):");
          System.out.println("8️⃣ - Reset search");
          System.out.println("9️⃣ - Back");
          System.out.print("Input option: ");
          int searchOption = s.nextInt();

          if (searchOption == 9) {
            // Exit search
            filterRunning = false;
            clearTerminal();
            continue;
          }
          else if (searchOption == 7) {
            // Print results
            listRunning = true;
            while (listRunning) { // Loop until user quits
              clearTerminal();
              search.refreshSearch(); // Make sure data is up to date.
              completeList(search.getClients(), s); // Show list of
filtered clients
            }
          }
          else if (searchOption == 6) {
            // Suspicious
            search.addFilter(5, "", false);
            clearTerminal();
          }
          else if (searchOption == 8) {
            // Reset search
            search.resetSearch();
            clearTerminal();
          }
          else {
            // Add filter where query is required
            clearTerminal();
            s.nextLine();
            System.out.print("Query:");
            String query = s.nextLine(); // Get query
            // Apply filter
            search.addFilter(searchOption - 1, query, false);
            clearTerminal();
          }
        }
      }
      else if (option == 4) {
        clientRunning = true;
        while (clientRunning) {
          System.out.println("Welcome new client, please select an
option:");
          System.out.println("1️⃣ - 🟢 - Start");
          System.out.println("2️⃣ - ❌ - Quit");
          System.out.print("Input option: ");
          int clientLoopOption = s.nextInt();
          clearTerminal();
```

```
          if (clientLoopOption == 1) {
            // Start client loop
            // Add client
            s.nextLine();
            System.out.print("Name: ");
            String name = s.nextLine(); // Get name
            System.out.print("Address: ");
            String address = s.nextLine(); // Get address
            System.out.print("Phone: ");
            String phone = s.nextLine(); // Get phone
            System.out.print("Email: ");
            String email = s.nextLine(); // Get email

            // Create new client with auto-generated ID
            // Populate with given details
            Client client = new Client();
            client.setName(name);
            client.setAddress(address);
            client.setPhone(phone);
            client.setEmail(email);
            client.setNote("Automated: This client was created by the
client loop.");
            client.save(); // Save client to file

            clearTerminal();
            System.out.println("✅ - Thanks for your details!");

          }
          else {
            // Quit all loops
            clientRunning = false;
            running = false;
          }

        }

      }
      else {
        // Exit program
        System.out.println("Exiting program..");
        running = false;
      }
    }
  }

  /**
   * Prints a list of clients.
   * @param clients
   * @param s
   */
  private void completeList(ArrayList<Client> clients, Scanner s) {
    System.out.println("Available options:");
    int i = 0; // Index of client
```

```
    for (Client client : clients) {
      i++; // Increment index
      System.out.println(i + ": " + client.getName()); // Print client
name
    }

    // Final option will be exit
    i++;
    System.out.println(i + ": Exit to menu");

    System.out.print("Input option: ");
    int searchResult = s.nextInt(); // Get input

    if (searchResult != i) { // If not exit
      clearTerminal();
      Client targetClient = clients.get(searchResult - 1); // Get client

      // Show client details
      System.out.println("Name: " + targetClient.getName());
      System.out.println("Address: " + targetClient.getAddress());
      System.out.println("Phone: " + targetClient.getPhone());
      System.out.println("Email: " + targetClient.getEmail());
      System.out.println("Note: " + targetClient.getNote());
      System.out.println("");

      System.out.println("Available options:");
      System.out.println("1️⃣ - 📝 - Edit name");
      System.out.println("2️⃣ - 📝 - Edit address");
      System.out.println("3️⃣ - 📝 - Edit phone");
      System.out.println("4️⃣ - 📝 - Edit email");
      System.out.println("5️⃣ - 📝 - Edit note");
      System.out.println("6️⃣ - 🚨 - Delete client");
      System.out.println("7️⃣ - ❌ - Exit to client list");

      int editOption = s.nextInt(); // Get input

      if (editOption == 1) {
        // Edit name
        s.nextLine();
        System.out.print("New name: ");
        String newName = s.nextLine();
        clearTerminal();
        // Set name and save
        targetClient.setName(newName);
        targetClient.save();
      }
      else if (editOption == 2) {
        s.nextLine();
        System.out.print("New address: ");
        String newAddress = s.nextLine();
        clearTerminal();
        // Set address and save
        targetClient.setAddress(newAddress);
        targetClient.save();
```

```
      }
      else if (editOption == 3) {
        s.nextLine();
        System.out.print("New phone: ");
        String newPhone = s.nextLine();
        clearTerminal();
        // Set phone and save
        targetClient.setPhone(newPhone);
        targetClient.save();
      }
      else if (editOption == 4) {
        s.nextLine();
        System.out.print("New email: ");
        String newEmail = s.nextLine();
        clearTerminal();
        // Set email and save
        targetClient.setEmail(newEmail);
        targetClient.save();
      }
      else if (editOption == 5) {
        s.nextLine();
        System.out.print("New note: ");
        String newNote = s.nextLine();
        clearTerminal();
        // Set note and save
        targetClient.setNote(newNote);
        targetClient.save();
      }
      else if (editOption == 6) {
        clearTerminal();
        // Delete client
        targetClient.delete();
      }
      else if (editOption == 7) {
        // Exit to client list
        clearTerminal();
      }
    }
    else {
      // Exit to menu
      listRunning = false;
      clearTerminal();
    }
  }

  /**
   * Flushes terminal
   */
  private void clearTerminal() {
    System.out.print("\033[H\033[2J");
    System.out.flush();
  }

}
```

Search.java

```
import java.io.File;
import java.io.FilenameFilter;
import java.util.ArrayList;

public class Search {

  // Fields
  private ArrayList<Client> clients = new ArrayList<Client>();
  private ArrayList<Integer> filtersIndexes = new ArrayList<Integer>();
  private ArrayList<String> filtersQueries = new ArrayList<String>();

  /**
   * Constructor for Search
   */
  public Search() {
    // Get list of all .txt files in default directory
    File[] files = new File(".").listFiles(new FilenameFilter() {
      public boolean accept(File dir, String name) {
        return name.toLowerCase().endsWith(".txt");
      }
    });

    // Reverse files array to simulate a stack
    for (int i = files.length - 1; i >= 0; i--) {
      // Get file name without extension
      File file = files[i];
      String id = file.getName().replace(".txt", "");
      // Make into an instance of Client
      Client client = new Client(id);
      clients.add(client);

      // Remove file from array to simulate stack
      files[i] = null;
    }
  }

  /**
   * Refresh search
   * precondition: none
   * postcondition: search is reset, and all clients are shown
   */
  public void resetSearch() {
    // Clear list of filters
    filtersIndexes.clear();
    filtersQueries.clear();

    // Refresh data without filters
    refreshSearch();
```

```
  }

  /**
   * Refresh search with filteresIndexes and filtersQueries
   * precondition: filtersIndexes and filtersQueries are set
   * postcondition: search is refreshed, and only clients that match
filters are shown
   */
  public void refreshSearch() {
    clients = new ArrayList<Client>(); // Clear clients

    // Get list of all .txt files in default directory
    File[] files = new File(".").listFiles(new FilenameFilter() {
      public boolean accept(File dir, String name) {
        return name.toLowerCase().endsWith(".txt");
      }
    });

    // Reverse files array to simulate a stack
    for (int i = files.length - 1; i >= 0; i--) {
      // Get file name without extension
      File file = files[i];
      String id = file.getName().replace(".txt", "");
      // Make into an instance of Client
      Client client = new Client(id);
      clients.add(client);

      // Remove file from array to simulate stack
      files[i] = null;
    }

    // Loop through filtersindexes
    for (int i = 0; i < filtersIndexes.size(); i++) {
      int filterIndex = filtersIndexes.get(i);
      String filterQuery = filtersQueries.get(i);

      // Apply the filter
      addFilter(filterIndex, filterQuery, true);
    }
  }

  /**
   * Add filter to search
   * @param filterIndex Index of filter (0 = name, 1 = address, 2 = phone,
3 = email, 4 = note, 5 = suspicious)
   * @param filterQuery Query of filter
   * @param refresh Whether to refresh search
   * precondition: filterIndex and filterQuery are set
   * postcondition: filter is added to search, and search is refreshed.
   */
  public void addFilter(int index, String filter, boolean skipFilterAdd) {
    ArrayList<Client> filteredClients = new ArrayList<Client>(); // List
of clients that match filter
    if (!skipFilterAdd) {
```

```
      filtersIndexes.add(index);
      filtersQueries.add(filter);
    }
    if (index == 0) {
      for (Client client : clients) { // Search on basis of name contains
        if (client.getName().toLowerCase().contains(filter.toLowerCase()))
{
          filteredClients.add(client);
        }
      }
    }
    else if (index == 1) {
      for (Client client : clients) { // Search on basis of address
contains
        if
(client.getAddress().toLowerCase().contains(filter.toLowerCase())) {
          filteredClients.add(client);
        }
      }
    }
    else if (index == 2) {
      for (Client client : clients) { // Search on basis of phone contains
        if
(client.getPhone().toLowerCase().contains(filter.toLowerCase())) {
          filteredClients.add(client);
        }
      }
    }
    else if (index == 3) {
      for (Client client : clients) { // Search on basis of email contains
        if
(client.getEmail().toLowerCase().contains(filter.toLowerCase())) {
          filteredClients.add(client);
        }
      }
    }
    else if (index == 4) {
      for (Client client : clients) { // Search on basis of note contains
        if (client.getNote().toLowerCase().contains(filter.toLowerCase()))
{
          filteredClients.add(client);
        }
      }
    }
    else if (index == 5) {
      // Only include badly formatted emails
      for (Client client : clients) {
        String email = client.getEmail();
        // Regex to check x@y where x,y are anything.
        if (!email.matches("^(.+)@(.+)$")) {
          filteredClients.add(client);
        }
      }
    }
```

```
    // Set contents of clients to filteredClients
    clients = new ArrayList<Client>();
    for (Client client : filteredClients) {
      clients.add(client);
    }
  }

  /**
   * Get list of clients
   */
  public ArrayList<Client> getClients() {
    return clients;
  }

  /**
   * Get count of filtered clients
   */
  public int getSize() {
    return clients.size();
  }
}
```