Input, Process, Output Summary:

- **INPUT**:
    - User's choice from the menu.
    - Input required for adding, searching, deleting, or updating contacts.
    - Reading from files.
- **PROCESS**:
    - Load contacts from a file.
    - Add, search, display, delete, update, sort, or save contacts.
    - Validate user input.
- **OUTPUT**:
    - Menu options.
    - Feedback for actions like adding, deleting, updating, or saving contacts.
    - Displaying all contacts or search results.

```
// Pseudocode for Main Phonebook Application

// Declare scanner object to read user input
INITIALIZE scanner

// Declare phonebook object to manage contacts
INITIALIZE phonebook

// Main method
PROCEDURE main

    // Load contacts from the file "contacts.txt" into phonebook
    // INPUT: Read contacts from file
    CALL phonebook.loadFromFile WITH "contacts.txt"

    // Infinite loop to display menu and handle user choices
    WHILE true DO

        // OUTPUT: Display the menu options to the user
        DISPLAY "Phonebook Application"
        DISPLAY "1. Add Contact"
        DISPLAY "2. Search Contact"
        DISPLAY "3. Display All Contacts"
        DISPLAY "4. Delete Contact"
        DISPLAY "5. Update Contact"
        DISPLAY "6. Sort Contacts"
        DISPLAY "7. Save Contacts to File"
        DISPLAY "8. Exit"

        // INPUT: Ask user to choose an option
        DISPLAY "Enter your choice: "

        // PROCESS: Read and validate user's input
        SET choice TO 0
        TRY
            // INPUT: Parse user input as integer
    SET choice TO PARSE scanner input AS integer
CATCH NumberFormatException
    // OUTPUT: Display invalid input message
    DISPLAY "Invalid input. Please enter a number."
    CONTINUE // Restart the loop if input is invalid

// PROCESS: Handle different user choices using switch-case
SWITCH choice DO

    CASE 1:
        // INPUT: User chooses to add a contact
        // PROCESS: Add contact to phonebook
        CALL AddContact.addContact WITH phonebook AND scanner
        // OUTPUT: Confirm contact is added
        BREAK

    CASE 2:
        // INPUT: User chooses to search for a contact
        // PROCESS: Search for a contact in phonebook
        CALL SearchContact.searchContact WITH phonebook AND scanner
        // OUTPUT: Display search results
        BREAK

    CASE 3:
        // INPUT: User chooses to display all contacts
        // PROCESS: Retrieve and display all contacts from phonebook
        CALL DisplayAllContacts.displayAllContacts WITH phonebook
        // OUTPUT: List all contacts
        BREAK

    CASE 4:
        // INPUT: User chooses to delete a contact
        // PROCESS: Delete contact from phonebook
        CALL DeleteContact.deleteContact WITH phonebook AND scanner
        // OUTPUT: Confirm contact is deleted
        BREAK

    CASE 5:
        // INPUT: User chooses to update a contact
        // PROCESS: Update existing contact in phonebook
        CALL UpdateContact.updateContact WITH phonebook AND scanner
        // OUTPUT: Confirm contact is updated
        BREAK

    CASE 6:
        // INPUT: User chooses to sort contacts
        // PROCESS: Sort contacts in phonebook
        CALL SortContacts.sortContacts WITH phonebook
        // OUTPUT: Confirm contacts are sorted
        BREAK

    CASE 7:
            // INPUT: User chooses to save contacts to file
            // PROCESS: Save contacts to "contacts.txt" file
            CALL SaveContactsToFile.saveContactsToFile WITH phonebook
            // OUTPUT: Confirm contacts are saved
            BREAK

        CASE 8:
            // INPUT: User chooses to exit the application
            // PROCESS: Exit the program
            CALL System.exit(0)

        DEFAULT:
            // OUTPUT: Display message for invalid choice
            DISPLAY "Invalid choice. Please try again."
      END SWITCH
   END WHILE
END PROCEDURE
```

- **INPUT**: Name and phone number of the contact.
- **PROCESS**: Create a new contact and add it to the phonebook.
- **OUTPUT**: Confirmation message after the contact is added.

```
// Pseudocode for AddContact


// PROCEDURE addContact

PROCEDURE addContact(phonebook, scanner)


   // INPUT: Ask for and read the contact's name

   DISPLAY "Enter name: "

   SET name TO READ scanner input


   // INPUT: Ask for and read the contact's phone number

   DISPLAY "Enter phone number: "

   SET phoneNumber TO READ scanner input


   // PROCESS: Add the new contact to the phonebook

   CALL phonebook.addContact WITH new Contact(name, phoneNumber)


   // OUTPUT: Confirm contact is added
```

```
    DISPLAY "Contact added successfully."

END PROCEDURE
```

- **INPUT**: User enters the name and phone number of the contact.
- **PROCESS**: A new contact object is created and added to the phonebook.
- **OUTPUT**: A success message is displayed to confirm the action.

```
// Pseudocode for AddContact

// PROCEDURE addContact
PROCEDURE addContact(phonebook, scanner)

    // INPUT: Prompt and read the contact's name
    DISPLAY "Enter name: "
    SET name TO READ scanner input

    // INPUT: Prompt and read the contact's phone number
    DISPLAY "Enter phone number: "
    SET phoneNumber TO READ scanner input

    // PROCESS: Create a new contact and add it to the phonebook
    CALL phonebook.addContact WITH new Contact(name, phoneNumber)

    // OUTPUT: Display confirmation message
    DISPLAY "Contact added successfully."

END PROCEDURE
```

Here is the pseudocode for the `DeleteContact` class:

```plaintext
// Pseudocode for DeleteContact

// PROCEDURE deleteContact
PROCEDURE deleteContact(phonebook, scanner)

    // INPUT: Prompt and read the name of the contact to delete
    DISPLAY "Enter name to delete: "
    SET name TO READ scanner input

    // PROCESS: Attempt to delete the contact from the phonebook
    IF phonebook.deleteContact(name) THEN
        // OUTPUT: If contact is found and deleted, display success message
        DISPLAY "Contact deleted successfully."
    ELSE
        // OUTPUT: If contact is not found, display not found message
        DISPLAY "Contact not found."
    END IF

END PROCEDURE
```

### Breakdown:
- **INPUT**: User inputs the name of the contact they wish to delete.

- **PROCESS**: The system attempts to delete the contact from the phonebook.
- **OUTPUT**: A success message if the contact is deleted, or a message if the contact is not found.

Here’s the pseudocode for the `DisplayAllContacts` class:

```plaintext
// Pseudocode for DisplayAllContacts

// PROCEDURE displayAllContacts
PROCEDURE displayAllContacts(phonebook)

    // PROCESS: Call the phonebook method to display all contacts
    CALL phonebook.displayAllContacts()

END PROCEDURE
```

### Breakdown:
- **INPUT**: None (no user input required).
- **PROCESS**: Retrieve and display all contacts from the phonebook.
- **OUTPUT**: Contacts are displayed through the phonebook's display method.

Here’s the pseudocode for the `SaveContactsToFile` class:

```plaintext
// Pseudocode for SaveContactsToFile

// PROCEDURE saveContactsToFile
PROCEDURE saveContactsToFile(phonebook)
```

```
    // PROCESS: Save the contacts to a file
    CALL phonebook.saveToFile("contacts.txt")

    // OUTPUT: Display confirmation message
    DISPLAY "Contacts saved to file."

END PROCEDURE
```

### Breakdown:
- **INPUT**: None (no user input required).
- **PROCESS**: Save the phonebook contacts to a specified file.
- **OUTPUT**: A confirmation message indicating that contacts have been saved successfully.

Here’s the pseudocode for the `SearchContact` class:

```plaintext
// Pseudocode for SearchContact

// PROCEDURE searchContact
PROCEDURE searchContact(phonebook, scanner)

    // INPUT: Prompt and read the name of the contact to search
    DISPLAY "Enter name to search: "
    SET name TO READ scanner input

    // PROCESS: Search for the contact in the phonebook
    SET contact TO CALL phonebook.searchContact(name)

    // OUTPUT: Display the contact details or not found message
    IF contact IS NOT NULL THEN
```

~~~
        DISPLAY "Contact found: " + contact
    ELSE
        DISPLAY "Contact not found."
    END IF

END PROCEDURE
```
~~~

### Breakdown:
- **INPUT**: User inputs the name of the contact they wish to search for.
- **PROCESS**: The system searches for the contact in the phonebook.
- **OUTPUT**: Displays the contact details if found or a not found message if it doesn't exist.

Here’s the pseudocode for the `SortContacts` class:

~~~
```plaintext
// Pseudocode for SortContacts

// PROCEDURE sortContacts
PROCEDURE sortContacts(phonebook)

    // PROCESS: Sort the contacts in the phonebook
    CALL phonebook.sortContacts()

    // OUTPUT: Display confirmation message
    DISPLAY "Contacts sorted."

END PROCEDURE
```
~~~

### Breakdown:

- **INPUT**: None (no user input required).
- **PROCESS**: The system sorts the contacts in the phonebook.
- **OUTPUT**: A confirmation message indicating that the contacts have been sorted successfully.

Here’s the pseudocode for the `UpdateContact` class:

```plaintext
// Pseudocode for UpdateContact

// PROCEDURE updateContact
PROCEDURE updateContact(phonebook, scanner)

    // INPUT: Prompt and read the name of the contact to update
    DISPLAY "Enter name to update: "
    SET name TO READ scanner input

    // INPUT: Prompt and read the new phone number
    DISPLAY "Enter new phone number: "
    SET newPhoneNumber TO READ scanner input

    // PROCESS: Attempt to update the contact's phone number
    IF phonebook.updateContact(name, newPhoneNumber) THEN
        // OUTPUT: If update is successful, display success message
        DISPLAY "Contact updated successfully."
    ELSE
        // OUTPUT: If contact is not found, display not found message
        DISPLAY "Contact not found."
    END IF

END PROCEDURE
```

### Breakdown:

- **INPUT**: User inputs the name of the contact and the new phone number.

- **PROCESS**: The system attempts to update the contact's phone number in the phonebook.

- **OUTPUT**: Displays a success message if the update is successful or a not found message if the contact doesn't exist.

Here’s the pseudocode for the `Phonebook` class:

```plaintext
// Pseudocode for Phonebook

// CLASS Phonebook
CLASS Phonebook
   // ATTRIBUTES
   PRIVATE contacts AS Queue OF Contact

   // CONSTRUCTOR
   CONSTRUCTOR Phonebook()
      INITIALIZE contacts AS empty LinkedList
   END CONSTRUCTOR

   // METHOD addContact
   METHOD addContact(contact AS Contact)
      ADD contact TO contacts
   END METHOD

   // METHOD searchContact
   METHOD searchContact(name AS String) RETURNS Contact
      FOR EACH contact IN contacts DO
```

```
        IF contact.getName() EQUALS name (ignoring case) THEN
            RETURN contact
        END IF
    END FOR
    RETURN NULL // Contact not found
END METHOD

// METHOD displayAllContacts
METHOD displayAllContacts()
    FOR EACH contact IN contacts DO
        DISPLAY contact
    END FOR
END METHOD

// METHOD deleteContact
METHOD deleteContact(name AS String) RETURNS Boolean
    SET contact TO searchContact(name)
    IF contact IS NOT NULL THEN
        REMOVE contact FROM contacts
        RETURN TRUE // Contact deleted
    END IF
    RETURN FALSE // Contact not found
END METHOD

// METHOD updateContact
METHOD updateContact(name AS String, newPhoneNumber AS String) RETURNS Boolean
    SET contact TO searchContact(name)
    IF contact IS NOT NULL THEN
        contact.setPhoneNumber(newPhoneNumber)
        RETURN TRUE // Contact updated
    END IF
    RETURN FALSE // Contact not found
```

```
END METHOD

// METHOD sortContacts
METHOD sortContacts()
   SET sortedList TO new ArrayList OF contacts
   CALL Collections.sort(sortedList)
   SET contacts TO new LinkedList OF sortedList
END METHOD

// METHOD saveToFile
METHOD saveToFile(filename AS String)
   TRY
      CREATE PrintWriter for filename
      FOR EACH contact IN contacts DO
         WRITE contact.getName() + "," + contact.getPhoneNumber() TO file
      END FOR
   CATCH IOException AS e
      DISPLAY "Error saving to file: " + e.getMessage()
   END TRY
END METHOD

// METHOD loadFromFile
METHOD loadFromFile(filename AS String)
   TRY
      CREATE BufferedReader for filename
      SET line TO NULL
      WHILE (line IS NOT NULL) DO
         line = READ line FROM file
         IF line IS NOT NULL THEN
            SET parts TO line.split(",")
            IF length of parts EQUALS 2 THEN
               CALL addContact(new Contact(parts[0], parts[1]))
```

```
          END IF
        END IF
      END WHILE
    CATCH IOException AS e
      DISPLAY "Error loading from file: " + e.getMessage()
    END TRY
  END METHOD
END CLASS
```

### Breakdown:

- **Input**:
  - Name and phone number from user for adding, updating, or searching contacts.
  - Filename for saving or loading contacts.

- **Process**:
  - Manage contacts through adding, searching, deleting, updating, sorting, saving, and loading from a file.

- **Output**:

- Display contacts, confirmation messages for operations, or error messages for file handling.

END!!