A REPORT OF FOUR WEEK TRAINING

at

ThinkNEXT Technologies Private Limited

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE AWARD

OF THE DEGREE OF

BACHELOR OF TECHNOLOGY

(Computer Science and Engineering)

JUNE – JULY, 2025

SUBMITTED BY:

Harshdeep

2302545

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GURU NANAK DEV ENGINEERING COLLEGE, LUDHIANA

(An Autonomous College under UGC Act)

Certificate by Company/Industry/Institute

Certificate

This is to certify that Mr. **Harshdeep** (University Roll No. **2302545**), student of B.Tech (Computer Science and Engineering) at Guru Nanak Dev Engineering College, Ludhiana, has successfully completed one-month industrial training at **ThinkNEXT Technologies Private Limited** during the period **23 June 2025 to 21 July 2025**. The candidate's project and training performance have been found satisfactory and meet the requirements for submission of this training report.

Authorized Signatory: _____

Designation: _____

Date: _____

Candidate's Declaration

I, **Harshdeep**, hereby declare that I have undertaken four week training at **ThinkNEXT Technologies Private Limited** during the period from **23 June 2025 to 21 July 2025** in partial fulfillment of the requirements for the award of the degree of B.Tech (Computer Science and Engineering) at Guru Nanak Dev Engineering College, Ludhiana. The work presented in this report is an authentic record of the training undertaken by me.

Signature of the Student

The one-month industrial training Viva–Voce Examination of _____ has been held on _____ and accepted.

Signature of Internal Examiner

Signature of External Examiner

Abstract

This report documents the four week industrial training completed at **ThinkNEXT Technologies Private Limited** from **23 June 2025** to **21 July 2025**. The training focused on strengthening Core Java concepts, including object-oriented programming, exception handling, file input/output, collections framework, and multithreading. While GUI topics using Swing were covered briefly for demonstration, the primary emphasis was on Java Core — its architecture, design practices, and real-world application.

The practical component of the training involved developing a mini-project named *Contact Book Application*. The application demonstrates modular design using classes, collections for efficient data handling, file-based persistence for storage, and basic concurrency where required for UI responsiveness. During the training, I practiced standard development workflows such as requirement analysis, design, coding, debugging, and testing. Key outcomes include improved code organization, error handling strategies, data persistence techniques, and an understanding of trade-offs among data structures.

This document contains theoretical background, detailed implementation steps, test cases, results, and conclusions. It also includes future extension ideas such as database integration, improved search indexing, and migration to more modern UI frameworks. The appendix contains the complete source code and sample outputs.

Acknowledgment

I would like to express my sincere gratitude to ThinkNEXT Technologies Private Limited for providing the opportunity to undergo industrial training. The environment and tasks assigned during the training were instrumental in applying academic concepts to real-world programming assignments.

I am thankful to the Department of Computer Science and Engineering, Guru Nanak Dev Engineering College, Ludhiana for supporting and encouraging industry exposure as part of the curriculum. Special thanks to colleagues and peers who provided feedback and helped in testing the application during its development. Finally, I acknowledge the support of my family and friends for their encouragement during the training period.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter introduces the background, motivation, and goals for the training, together with a deep dive into the Java ecosystem and the role of Core Java in modern software engineering.

1.1 Background and Motivation

The rapid growth of software development across all sectors — from banking and education to healthcare and logistics — has made programming skills one of the most essential technical abilities in the modern world. Among the various programming languages available today, Java continues to hold a dominant position due to its reliability, portability, and rich ecosystem. Many organizations rely on Java-based systems for their day-to-day operations, from back-end services to desktop tools and enterprise-scale web applications.

The motivation behind selecting Core Java as the focus of this industrial training stemmed from its deep relevance in both academic and professional contexts. Although students often study Java as part of their curriculum, actual implementation in an industrial environment introduces a new layer of complexity that cannot be replicated in classroom learning. This training provided a platform to connect theoretical knowledge with practical applications such as problem-solving, debugging, and performance optimization.

Furthermore, the training aimed to strengthen the understanding of Object-Oriented Programming (OOP) concepts, which form the backbone of most modern programming languages. Mas-

tering Java fundamentals also helps in learning advanced technologies such as Spring Boot, Hibernate, Android development, and cloud-based Java frameworks. These skills open a wide range of career opportunities in software development, application engineering, and research-oriented computing.

1.2 Java: A Brief History and Position

Java was introduced by Sun Microsystems in 1995 with the intent of creating a platform-independent language for consumer electronics and networked devices. Over the decades Java evolved into a mainstream language powering enterprise servers, Android apps, desktop tools, and large-scale systems. Major milestones include the introduction of the JVM, the release of standardized editions (SE, EE, ME), and significant improvements in performance, concurrency utilities, and language features.

1.3 Java Architecture and JVM Internals

Java's architecture is based on the compilation of source code into bytecode which runs on the Java Virtual Machine (JVM). The JVM abstracts hardware and OS differences, providing a stable runtime environment. Key components of the JVM and runtime behavior covered during training include:

- Class loader subsystem and delegation model.
- Bytecode verifier for security and correctness.
- Runtime data areas: method area, heap, stack, program counter, native method stacks.
- Garbage collection algorithms (generational GC, CMS, G1) and tuning basics.

Understanding JVM internals helps developers write more memory-efficient and performant

applications, by reducing unnecessary object creation, managing references properly, and understanding how the GC affects application throughput and latency.

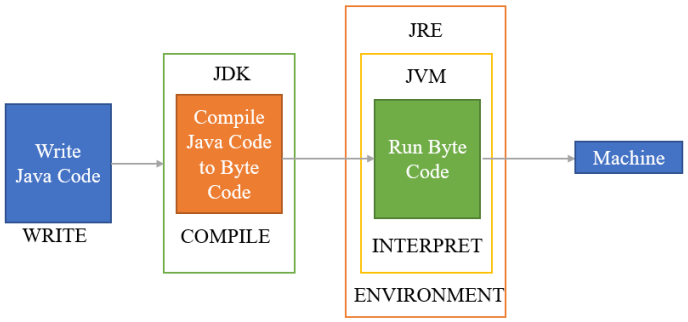

Figure 1.1. Java Platform Architecture

1.4 Java Language Features and Advantages

- Strong typing and compile-time checks that reduce runtime errors.
- Rich standard library (I/O, networking, concurrency, collection classes).
- Mature tooling (IDEs like IntelliJ/Eclipse/NetBeans, build tools such as Maven and Gradle).
- Backward compatibility across versions—making long-term maintenance easier.

1.5 Typical Java Development Workflow

The practical workflow used during the training:

1. Requirement gathering and design (UML sketches, class responsibilities).
2. Coding using a standard code style and following OOP principles.
3. Unit testing and simple test-driven approaches where applicable.
4. Debugging, logging, and iterative refinement.
5. Packaging and simple deployment (jar files).

1.6 Comparison with Other Languages

Programming languages have evolved over decades to cater to different development needs, from system-level programming to high-level application design. Among these, Java occupies a unique middle ground, balancing ease of use with strong performance and a rich ecosystem. Comparing Java with other prominent programming languages provides a better understanding of its continued dominance in industry and education.

1.6.1 Java vs. C and C++

Java was developed with the intention of overcoming several limitations of C and C++. While C and C++ offer direct memory management through pointers and manual allocation, they also introduce significant risks such as memory leaks, segmentation faults, and dangling pointers. Java eliminates these hazards through automatic garbage collection and strict memory safety checks. Moreover, C and C++ are compiled into platform-dependent binaries, meaning programs must be rebuilt separately for different operating systems. Java, however, compiles into platform-independent bytecode, which the JVM interprets or compiles at runtime, ensuring that a single executable can run on multiple platforms without modification. This makes Java more portable and cost-effective for large-scale deployments.

Figure 1.2. Java vs C/C++

1.6.2 Java vs. Python

Python has gained immense popularity in recent years due to its simplicity, concise syntax, and strong support for rapid prototyping, data analysis, and machine learning. However, Java still maintains an edge in large-scale enterprise systems. Java's static typing allows for early detection of type-related errors at compile time, which is particularly valuable in maintaining and scaling large codebases. Python, being dynamically typed, can introduce runtime errors that might not surface until a specific code path is executed. Performance-wise, Java generally runs faster because of its Just-In-Time (JIT) compiler, which converts frequently executed bytecode into optimized machine code. Python relies on an interpreter, which makes it slower in execution speed for computationally intensive applications. That said, Python offers ease of learning and faster development cycles, whereas Java ensures robustness, maintainability, and predictable performance, especially for multithreaded or networked applications.

Figure 1.3. Java vs Python

1.6.3 Java vs. JavaScript

Despite the similarity in their names, Java and JavaScript are fundamentally different. Java is a compiled, strongly typed, class-based language, while JavaScript is interpreted, dynamically typed, and prototype-based. JavaScript primarily operates within web browsers and is event-driven, whereas Java runs on the JVM and is used for backend systems, Android apps, and large-scale business applications.

In modern development, these two languages often coexist — Java may power the backend logic while JavaScript (or its frameworks like React or Angular) manages the front-end interface. The interoperability between Java-based REST APIs and JavaScript-based clients showcases how both languages are vital to full-stack development.

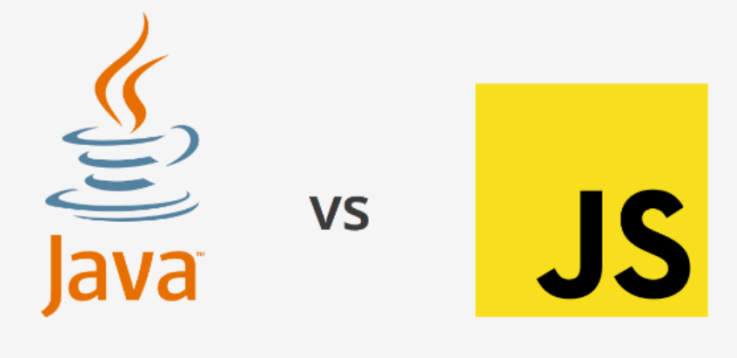

Figure 1.4. Java vs JavaScript

1.7 Learning Outcomes From Introduction

By the end of the introductory sessions, I was comfortable with:

- Java toolchain (JDK, javac, java, and key IDE features).
- Reading and interpreting stack traces and JVM logs.
- Writing modular code with clear class responsibilities.

Chapter 2

Training Work Undertaken

This chapter describes in detail the sequence of training topics, hands-on exercises, and the mini-project implementation.

2.1 Training Schedule Recap

Week	Coverage / Activities
Week 1	Java Basics: Syntax, Data types, Operators, Control flow (if, switch, loops), arrays and Strings. Practical exercises and simple console programs.
Week 2	OOP deep-dive: Classes, objects, constructors, inheritance, polymorphism, interfaces, and encapsulation with design exercises.
Week 3	Exception handling, file I/O (text and binary), serialization, and Collections framework (ArrayList, HashMap, HashSet).
Week 4	Multithreading basics, swing, design and development of Contact Book Application, testing, and documentation.

2.2 Java Basics - Exercises and Notes

During hands-on exercises I implemented:

- Programs to practice primitive types, overflow cases and numeric casting.

- String manipulation tasks (splitting, searching, pattern matching).
- Array algorithms (sorting, searching, merging).

2.2.1 Sample: Numeric Casting and Rounding

```
1 double d = 9.78;
2 int i = (int) d; // truncates to 9
3 System.out.println("Truncated: "+i);
```

Listing 2.1: Numeric Casting Example

2.3 OOP: Design and Implementation

A large portion of the training reinforced object oriented design principles and how to decompose an application into classes.

2.3.1 Class Responsibilities

In the Contact Book project classes were organized into:

- Model classes (Contact) — storing data fields and accessors.
- Manager classes (ContactManager) — providing business logic and persistence.
- UI classes (ContactBookApp) — presenting data and accepting user input (Swing).

2.3.2 Interfaces and Abstraction

Interfaces were used to describe search and persistence contracts, allowing swapping of persistence mechanisms (file vs. DB) without changing business logic.

2.4 Exception Handling Deep Dive

We studied:

- Checked vs unchecked exceptions and trade-offs in API design.
- Creating custom exceptions (e.g., ContactNotFoundException).
- Best practices: fail-fast, meaningful messages, and resource cleanup with try-with-resources.

2.5 File Handling and Persistence

Methods practiced:

- Plain text CSV-style persistence for readability.
- Binary serialization for compact storage using ObjectOutputStream (with Serializable).
- Handling file-not-found and permission issues robustly.

2.5.1 Sample: Writing contacts to CSV

```
1 try(BufferedWriter bw = new BufferedWriter(new FileWriter("contacts
     .csv"))){
2     for(Contact c : contacts){
3         bw.write(c.getName()+","+c.getPhone()+","+c.getEmail());
4         bw.newLine();
5     }
6 }
```

Listing 2.2: Sample CSV Save

2.6 Collections and Data Structures

Detailed exercises included:

- Choosing between ArrayList and LinkedList for insertion vs traversal workloads.
- Using HashMap for phone-to-contact lookup (O(1) average-time).
- Using TreeMap for sorted retrievals (log n).

2.7 Multithreading and Concurrency

Hands-on topics:

- Creating threads through Runnable and Thread classes.
- Synchronization primitives (synchronized, volatile).
- Simple thread-safe collections and atomic variables.
- Avoiding deadlocks and designing thread-safe operations.

2.7.1 Sample: Simple Thread with Runnable

```
Runnable r = () -> {
    System.out.println("Background task started.");
    // do work
};
new Thread(r).start();
```

Listing 2.3: Runnable Example

2.8 Mini-Project: Contact Book Application — Design

2.8.1 Requirements

- Add, view, edit and remove contacts (Name, Phone, Email).
- Search by name or phone number.
- Persist data between runs (file-based).
- Prevent duplicate phone numbers.
- Basic Swing interface for demonstration.

2.8.2 High-Level Design

- **Contact.java:** data model for a contact.
- **ContactManager.java:** implements CRUD operations, indexing, save/load.
- **ContactBookApp.java:** GUI and main application flow (external file referenced in Appendix).

2.9 Implementation Notes and Challenges

During implementation, common issues included input validation, handling of newline characters in files, and race conditions when saving during UI operations. To address these:

- Validated phone numbers using regex before insertion.
- Used try-with-resources to ensure streams are closed.
- Separated UI event thread from long-running I/O by spinning a worker thread for save/load tasks.

Chapter 3

Results and Discussion

This chapter presents final test results, sample outputs and discussion of design trade-offs.

3.1 Functional Test Cases

The application was tested with a set of representative scenarios:

Table 3.1. Functional Test Results

Test Case	**Description**	**Result**
Add New Contact	Verify new entry is stored and visible.	Pass
Add Duplicate Contact	Enter existing phone number.	Fail (handled)
Search Contact	Retrieve existing contact.	Pass
Delete Contact	Remove by phone number.	Pass
Save/Load File	Verify data saved and reloaded.	Pass

3.2 Performance and Observations

- For small datasets (under 10k contacts), file-based persistence is adequate.
- HashMap indexing ensures constant time lookup by phone.
- Name-based search uses linear scanning (OK for small datasets; indexing needed for larger

data).

- Swing UI responsiveness was improved by delegating file I/O off the Event Dispatch Thread (EDT).

3.3 Sample Screenshots

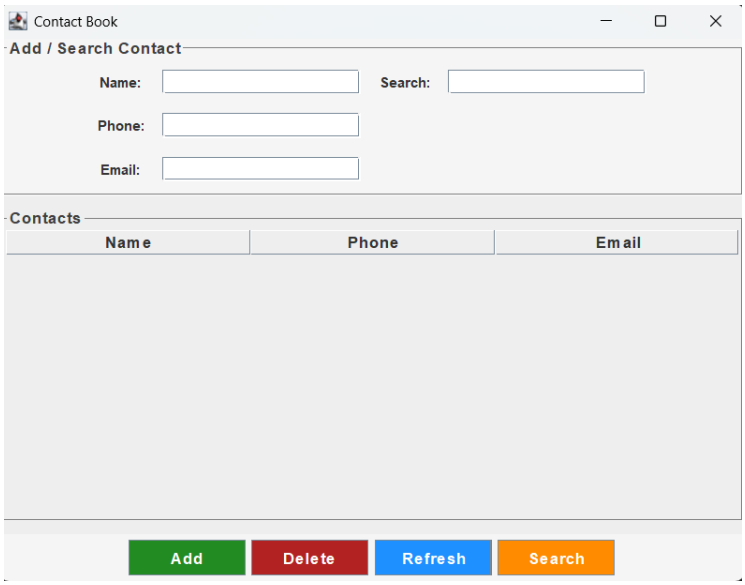

Figure 3.1. Main Window — Contact Book

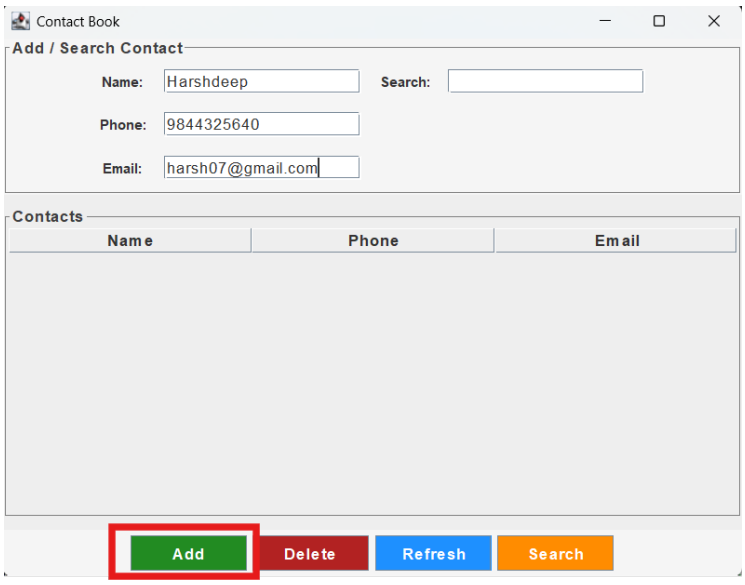

Figure 3.2. Add Contact 1

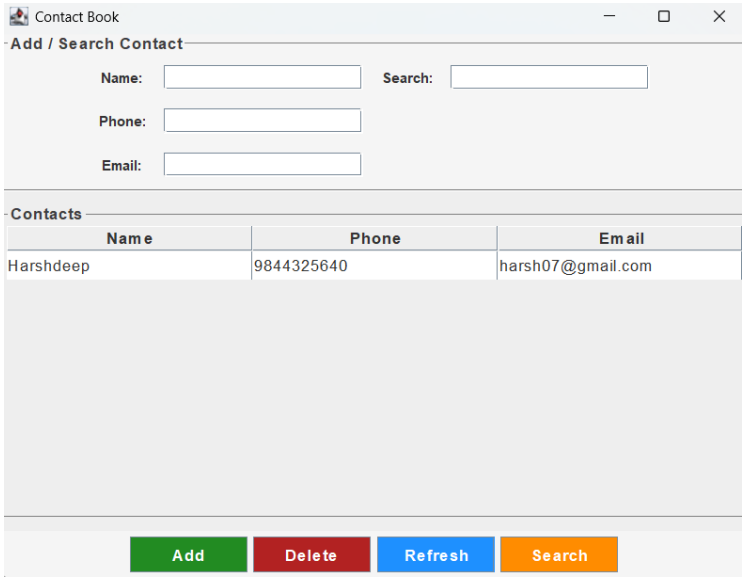

Figure 3.3. Add Contact 2

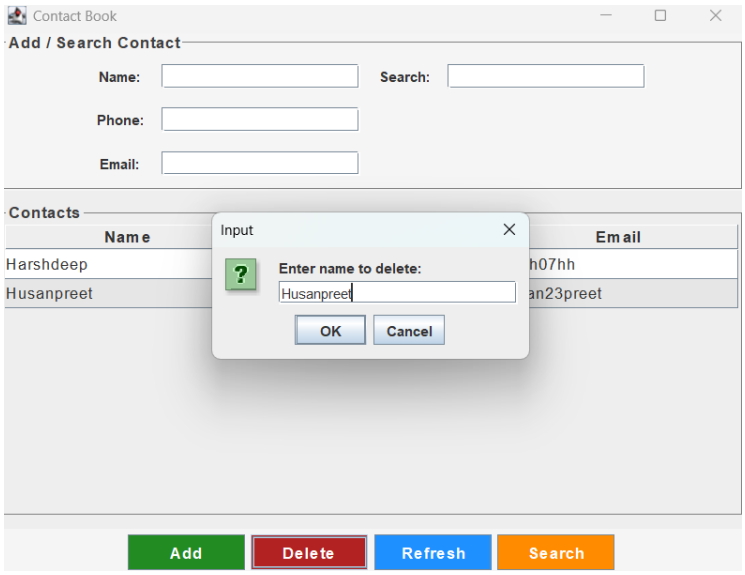

Figure 3.4. Delete Contact 1

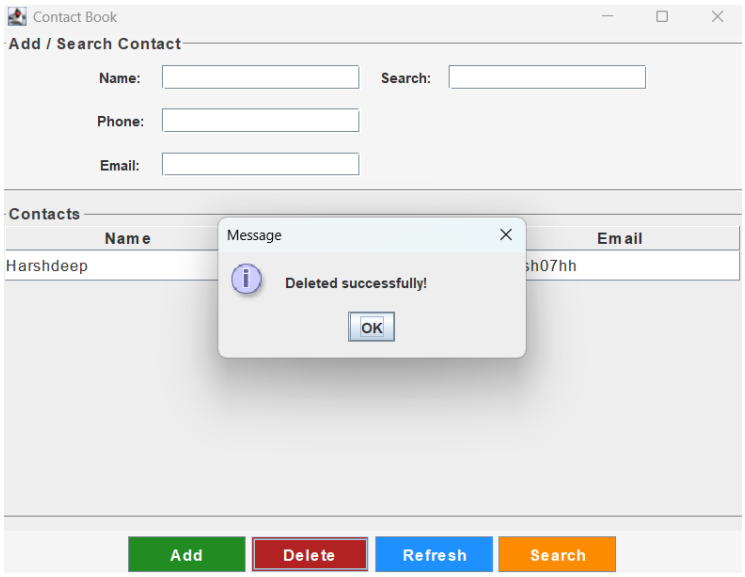

Figure 3.5. Delete Contact 2

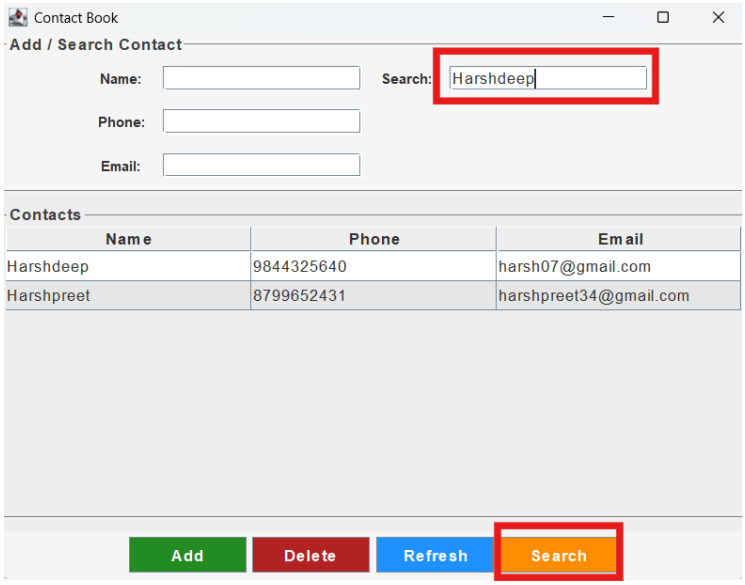

Figure 3.6. Search Contact 1

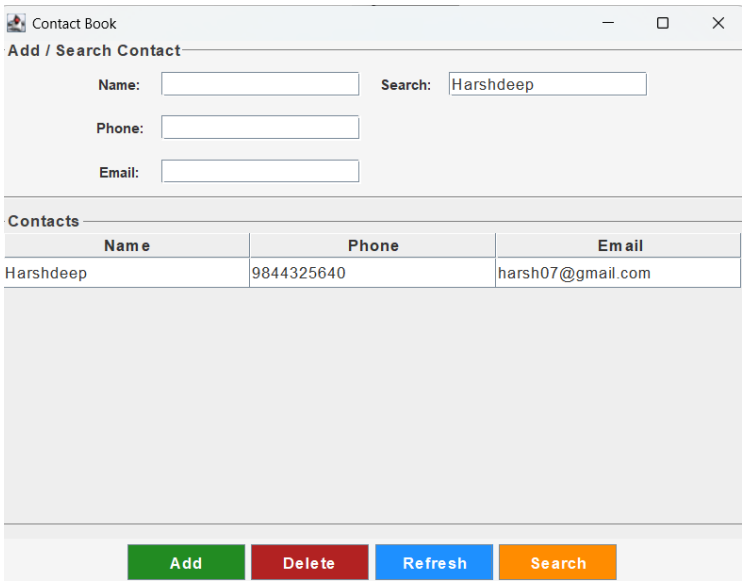

Figure 3.7. Search Contact 2

3.4 Quality Attributes and Testing

The project concentrated on:

- **Correctness:** Business logic (no duplicates, correct saves/loads).
- **Reliability:** Proper exception handling during I/O.
- **Usability:** Simple, clear UI flows for primary tasks.

Chapter 4

Conclusion and Future Scope

4.1 Conclusion

The four week industrial training at ThinkNEXT Technologies Private Limited provided robust, hands-on experience in Core Java development. By working on the Contact Book Application, I applied object-oriented design patterns, used the Collections Framework effectively, and implemented file-based persistence. I also acquired practical skills in debugging, exception handling, thread design, and maintaining separation between UI and business logic. The experience solidified my understanding of how foundational Java knowledge applies directly to real-world software projects.

4.2 Future Scope

This project has a range of natural extensions. Major next steps include:

- **Database Integration:** Move from file persistence to relational database (MySQL/SQLite) for scalability and query support.
- **Advanced Search Indexing:** Implement prefix trees or inverted indices for efficient name lookups in large datasets.
- **REST API Layer:** Expose contact operations through a RESTful API, enabling web/mobile front-ends.

- **Improved UI:** Migrate to JavaFX or a web UI for modern look-and-feel and improved UX.
- **Authentication & Authorization:** Add user roles and secure endpoints to support multiple users and permissions.
- **Background Sync:** Implement scheduled background sync and backups to cloud storage.

These improvements would make the system more production-ready and suitable for institutional deployment.

Bibliography

[1] J. Gosling, B. Joy, G. Steele, and G. Bracha, *The Java Language Specification*, Addison-Wesley, 2015.

[2] J. Bloch, *Effective Java*, 3rd ed., Addison-Wesley, 2018.

[3] K. Sierra and B. Bates, *Head First Java*, 2nd ed., O'Reilly Media, 2005.

[4] H. Schildt, *Java: The Complete Reference*, 11th ed., McGraw-Hill Education, 2022.

[5] Oracle Corporation, "Java SE Documentation and Tutorials," *Online Resource*, Available: `https://docs.oracle.com/javase/`

Chapter A

Appendix A: ContactBookApp.java (External file)

```
1  package contactbook;
2
3  import javax.swing.*;
4  import javax.swing.border.*;
5  import javax.swing.table.*;
6  import java.awt.*;
7  import java.awt.event.*;
8  import java.io.*;
9  import java.util.*;
10
11 class Contact {
12     String name, phone, email;
13     Contact(String n, String p, String e) { name = n; phone = p;
           email = e; }
14     @Override
15     public String toString() { return name + "," + phone + "," +
           email; }
16 }
17
```

```
18 public class ContactBookApp extends JFrame {
19     private JTextField nameField, phoneField, emailField,
          searchField;
20     private JTable contactTable;
21     private DefaultTableModel tableModel;
22     private java.util.List<Contact> contacts;
23     private File file;
24
25     public ContactBookApp() {
26         setTitle("Contact Book");
27         setSize(650, 500);
28         setDefaultCloseOperation(EXIT_ON_CLOSE);
29         setLocationRelativeTo(null);
30         setLayout(new BorderLayout(10, 10));
31         contacts = new ArrayList<>();
32         file = new File("contacts.txt");
33         loadContacts();
34
35         // Top Panel - Inputs + Search
36         JPanel topPanel = new JPanel(new GridBagLayout());
37         topPanel.setBorder(new TitledBorder(BorderFactory.
              createLineBorder(Color.GRAY), "Add / Search Contact",
              TitledBorder.LEADING, TitledBorder.TOP, new Font("Arial"
              , Font.BOLD, 14), Color.DARK_GRAY));
38         topPanel.setBackground(new Color(245, 245, 245));
39         GridBagConstraints c = new GridBagConstraints();
40         c.insets = new Insets(8, 8, 8, 8);
```

```
41
42          Font labelFont = new Font("Arial", Font.BOLD, 14);
43          Font fieldFont = new Font("Arial", Font.PLAIN, 14);
44
45          c.gridx = 0; c.gridy = 0; topPanel.add(new JLabel("Name:"),
                c);
46          c.gridx = 1; nameField = new JTextField(15); nameField.
               setFont(fieldFont); topPanel.add(nameField, c);
47
48          c.gridx = 0; c.gridy = 1; topPanel.add(new JLabel("Phone:")
               , c);
49          c.gridx = 1; phoneField = new JTextField(15); phoneField.
               setFont(fieldFont); topPanel.add(phoneField, c);
50
51          c.gridx = 0; c.gridy = 2; topPanel.add(new JLabel("Email:")
               , c);
52          c.gridx = 1; emailField = new JTextField(15); emailField.
               setFont(fieldFont); topPanel.add(emailField, c);
53
54          c.gridx = 2; c.gridy = 0; topPanel.add(new JLabel("Search:"
               ), c);
55          c.gridx = 3; searchField = new JTextField(15); searchField.
               setFont(fieldFont); topPanel.add(searchField, c);
56
57          add(topPanel, BorderLayout.NORTH);
58
59          // Center Panel - Table
```

```
60          tableModel = new DefaultTableModel(new Object[]{"Name", "
               Phone", "Email"}, 0);
61          contactTable = new JTable(tableModel) {
62              public Component prepareRenderer(TableCellRenderer
                   renderer, int row, int column) {
63                  Component c = super.prepareRenderer(renderer, row,
                       column);
64                  if (!isRowSelected(row)) {
65                      c.setBackground(row % 2 == 0 ? Color.WHITE :
                           new Color(230, 230, 230));
66                  }
67                  return c;
68              }
69          };
70          contactTable.setRowHeight(25);
71          JTableHeader header = contactTable.getTableHeader();
72          header.setFont(new Font("Arial", Font.BOLD, 14));
73          contactTable.setFont(new Font("Arial", Font.PLAIN, 14));
74          contactTable.setEnabled(false);
75
76          JScrollPane tableScroll = new JScrollPane(contactTable);
77          tableScroll.setBorder(new TitledBorder(BorderFactory.
               createLineBorder(Color.GRAY), "Contacts", TitledBorder.
               LEADING, TitledBorder.TOP, new Font("Arial", Font.BOLD,
               14), Color.DARK_GRAY));
78          add(tableScroll, BorderLayout.CENTER);
79
```

```
80          // Bottom Panel - Buttons
81          JPanel buttonPanel = new JPanel(new FlowLayout());
82          buttonPanel.setBackground(new Color(245, 245, 245));
83
84          JButton addButton = createButton("Add", new Color(34, 139,
                34));
85          JButton deleteButton = createButton("Delete", new Color
                (178, 34, 34));
86          JButton refreshButton = createButton("Refresh", new Color
                (30, 144, 255));
87          JButton searchButton = createButton("Search", new Color
                (255, 140, 0));
88
89          buttonPanel.add(addButton); buttonPanel.add(deleteButton);
                buttonPanel.add(refreshButton); buttonPanel.add(
                searchButton);
90          add(buttonPanel, BorderLayout.SOUTH);
91
92          // Button Actions
93          addButton.addActionListener(e -> addContact());
94          deleteButton.addActionListener(e -> deleteContact());
95          refreshButton.addActionListener(e -> displayContacts());
96          searchButton.addActionListener(e -> searchContacts());
97
98          displayContacts();
99          setVisible(true);
100     }
```

```
101
102     // Create styled buttons
103     private JButton createButton(String text, Color bg) {
104         JButton button = new JButton(text);
105         button.setBackground(bg); button.setForeground(Color.WHITE)
                ;
106         button.setFont(new Font("Arial", Font.BOLD, 14));
107         button.setFocusPainted(false);
108         button.setPreferredSize(new Dimension(100, 30));
109         return button;
110     }
111
112     private void addContact() {
113         String name = nameField.getText().trim();
114         String phone = phoneField.getText().trim();
115         String email = emailField.getText().trim();
116         if (name.isEmpty() || phone.isEmpty() || email.isEmpty()) {
117             JOptionPane.showMessageDialog(this, "Please fill all
                    fields!"); return;
118         }
119         contacts.add(new Contact(name, phone, email));
120         saveContacts(); displayContacts();
121         nameField.setText(""); phoneField.setText(""); emailField.
                setText("");
122     }
123
124     private void deleteContact() {
```

```
125         String nameToDelete = JOptionPane.showInputDialog(this, "
               Enter name to delete:");
126         if (nameToDelete == null || nameToDelete.trim().isEmpty())
               return;
127         boolean found = false;
128         Iterator<Contact> it = contacts.iterator();
129         while (it.hasNext()) {
130             Contact c = it.next();
131             if (c.name.equalsIgnoreCase(nameToDelete.trim())) { it.
                   remove(); found = true; break; }
132         }
133         if (found) { saveContacts(); displayContacts(); JOptionPane
               .showMessageDialog(this, "Deleted successfully!"); }
134         else JOptionPane.showMessageDialog(this, "Contact not found
               !");
135     }
136
137     private void searchContacts() {
138         String query = searchField.getText().trim().toLowerCase();
139         tableModel.setRowCount(0);
140         for (Contact c : contacts) {
141             if (c.name.toLowerCase().contains(query)) {
142                 tableModel.addRow(new Object[]{c.name, c.phone, c.
                       email});
143             }
144         }
145     }
```

```
146
147     private void displayContacts() {
148         tableModel.setRowCount(0);
149         for (Contact c : contacts) tableModel.addRow(new Object[]{c
                .name, c.phone, c.email});
150     }
151
152     private void loadContacts() {
153         if (!file.exists()) return;
154         try (BufferedReader br = new BufferedReader(new FileReader(
                file))) {
155             String line;
156             while ((line = br.readLine()) != null) {
157                 String[] parts = line.split(",");
158                 if (parts.length == 3) contacts.add(new Contact(
                        parts[0], parts[1], parts[2]));
159             }
160         } catch (IOException e) { e.printStackTrace(); }
161     }
162
163     private void saveContacts() {
164         try (BufferedWriter bw = new BufferedWriter(new FileWriter(
                file))) {
165             for (Contact c : contacts) { bw.write(c.toString()); bw
                    .newLine(); }
166         } catch (IOException e) { e.printStackTrace(); }
167     }
```

```
168
169     public static void main(String[] args) { new ContactBookApp();
            }
170 }
```

Listing A.1: ContactBookApp.java — Complete Source (external file)