VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELAGAVI 590018

Project Report on

"VISION DOTS"

By

MANIKANTA KARADIGUDDA (1BM24CS161)
MANISH NAYAKA R (1BM24CS162) MANISHA C S (1BM24CS163)
MANJU K S(1BM24CS164)

Under the Guidance of

MONISHA H M
Assistant Professor, Department of CSE
BMS College of Engineering

Work carried out at

Department of Computer Science and Engineering
BMS College of Engineering (Autonomous college under VTU)
P.O. Box No.: 1908, Bull Temple Road, Bangalore-560 019
2025-2026

BMS COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the OOPS with JAVA project titled " **VISION DOTS"** has been carried out by MANIKANTA KARADIGUDDA (1BM24CS161), MANISH NAYAKA R (1BM24CS162), MANISHA C S (1BM24CS163) , MANJU K S(1BM24CS164) during the academic year 2025-2026.

Signature of the guide
MONISHA H M
Assistant Professor,
Department of Computer Science and Engineering
BMS College of Engineering, Bangalore

BMS COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

We, MANIKANTA KARADIGUDDA (1BM24CS161), MANISH NAYAKA R (1BM24CS162)MANISHA.C.S (1BM24CS163) , MANJU K S(1BM24CS164) students of 3rd Semester, B.E, Department of Computer Science and Engineering, BMS College of Engineering, Bangalore, hereby declare that, this project work entitled " **VISION DOTS** " has been carried out by us under the guidance of MONISHA H M , Assistant Professor, Department of CSE, BMS College of Engineering, Bangalore during the academic semester Sep-Dec 2025. We also declare that to the best of our knowledge and belief, the project reported here is not from part of any other report by any other students.

Signature of the Candidates

MANIKANTA KARADIGUDDA (1BM24CS161)

MANISH NAYAKA R (1BM24CS162)

MANISHA C S (1BM24CS163)

MANJU K S(1BM24CS164)

TABLE OF CONTENTS

Problem Statement

Project Title

Vision Dots: Text to Braille Conversion System

Problem Definition

To develop a Java-based application that converts English text (including alphabets, numbers, and punctuation) into Braille representation. The system should:

- Accept plain text input from users through a graphical interface
- Process the text considering:
- Uppercase letters (require capital sign prefix)
- Numbers (require number sign prefix)
- Special characters and punctuation
- Convert each character into corresponding Braille dot patterns
- Format the output into a three-row representation for easy visualization
- Display the Braille output in a user-friendly GUI
- Handle exceptions for unsupported characters gracefully

Objectives

To implement Braille conversion rules accurately.

To demonstrate object-oriented programming principles.

To create an intuitive graphical user interface.

To provide educational value for understanding Braille system.

Introduction

What is Braille?

Braille is a tactile writing system used by people who are visually impaired. It was invented by Louis Braille in 1824. The system uses patterns of raised dots arranged in cells of up to six dots in a 3×2 configuration. Each pattern represents a character, numeral, or punctuation mark.

Digital Braille Representation

In digital systems, Braille dots are often represented using binary notation:

Dot Positions:	Standard Numbering:
● ●	1 4
● ●	2 5
● ●	3 6

A "1" indicates a raised dot, and "0" indicates a flat position. For example, "100000" represents the letter 'a' (only dot 1 raised).

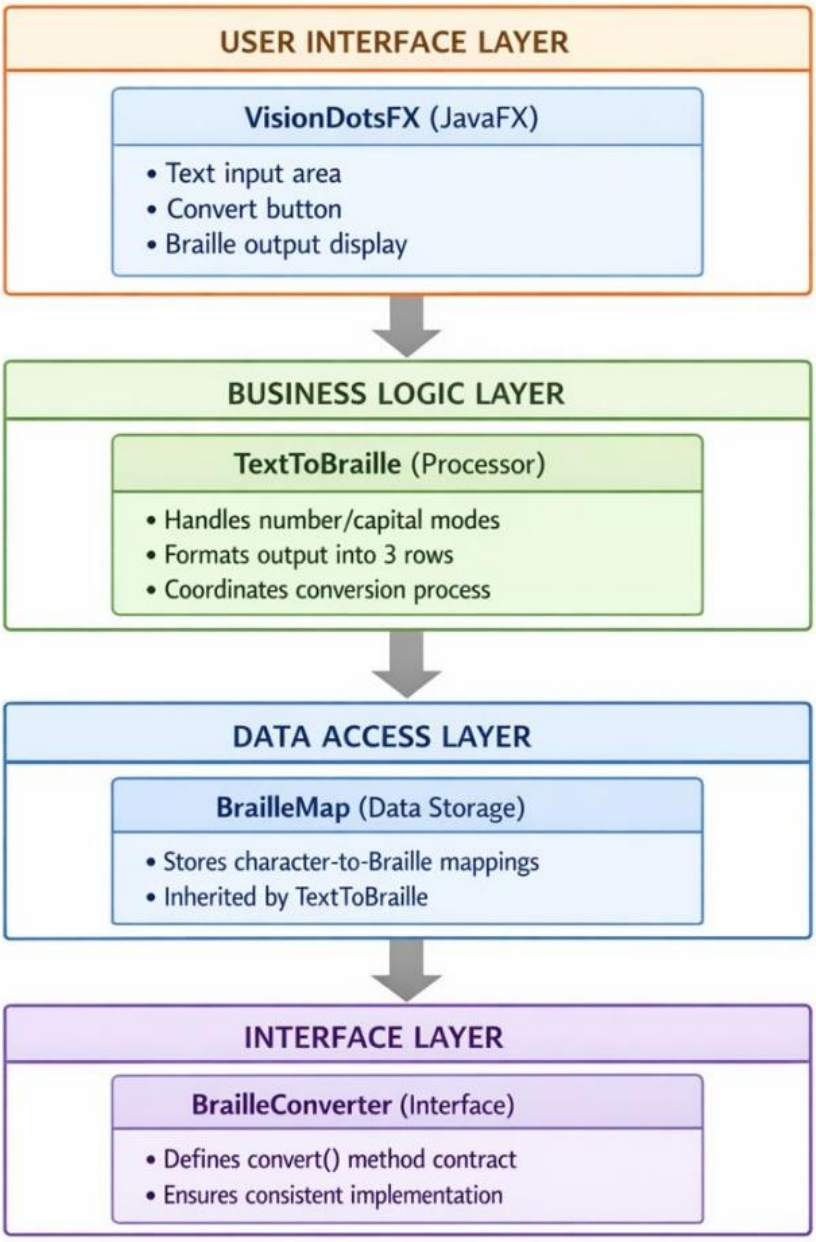
Project Significance

This project bridges the gap between digital text and Braille representation, serving as:

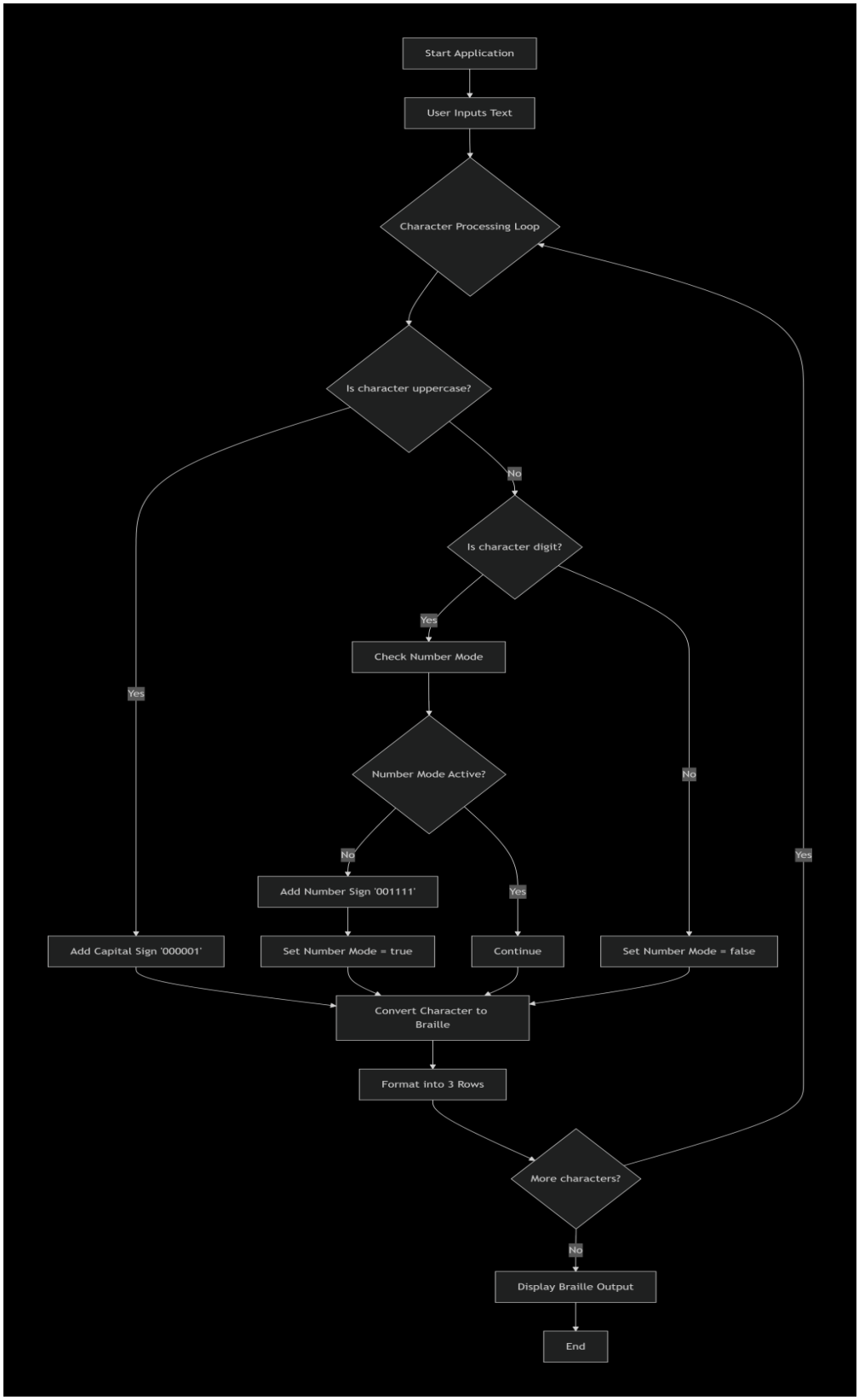
- An educational tool for learning Braille
- A foundation for more advanced accessibility tools
- A demonstration of Java OOP capabilities in real-world applications

Overview

System Architecture



Flow Diagram



Tools Used

Programming Languages and Frameworks

Java SE 17+: Core programming language

JavaFX 25.0.1: For graphical user interface

Java Collections Framework: HashMap for data storage

Development Environment

IDE: Visual Studio Code / IntelliJ IDEA / Eclipse

Build Tool: Standard Java Compiler (javac)

Version Control: Git (optional)

Required Libraries

JavaFX SDK (for GUI components)

Java Runtime Environment

Configuration Files

launch.json: VS Code debugging configuration with JavaFX module path

OOPs concept used & it's Explanation

Abstraction

```
public interface BrailleConverter {
    String convert(char ch) throws InvalidSymbolException;
}
```

Concept: Hiding implementation details and showing only essential features

Implementation: BrailleConverter interface defines what a converter should do without specifying how

Benefit: Allows different Braille implementations while maintaining the same interface

Encapsulation

```
public class BrailleMap implements BrailleConverter {
    protected HashMap<Character, String> braille = new HashMap<>();
    // ... private loading method
}
```

Concept: Bundling data and methods that operate on that data within a single unit

Implementation:

- braille map is protected (accessible to subclasses only)
- loadBraille() is private (internal implementation detail)
- Public convert() method provides controlled access

Benefit: Data integrity and controlled modification

Inheritance

```
public class TextToBraille extends BrailleMap {
    // Inherits braille mapping and convert method
}
```

Concept: Creating new classes based on existing classes

Implementation: TextToBraille extends BrailleMap, inheriting:

- Braille character mappings
- convert() method
- Data structure initialization

Benefit: Code reusability and hierarchical organization

Polymorphism

```
BrailleConverter converter = new BrailleMap();
String dots = converter.convert('a');
```

Concept: One interface, multiple implementations

Implementation:

- Method overriding in BrailleMap
- Interface-based programming
- Runtime method resolution

Benefit: Flexible and extensible code structure

Exception Handling

```
public class InvalidSymbolException extends Exception {
    public InvalidSymbolException(String message) {
        super(message);
    }
}
```

Concept: Handling runtime errors gracefully

Implementation: Custom checked exception for invalid characters

Benefit: Robust error handling and user feedback

Implementation

BrailleConverter Interface

```
package braille;

public interface BrailleConverter {
    // Contract method: Convert a character to Braille representation
    // Throws InvalidSymbolException for unsupported characters
    String convert(char ch) throws InvalidSymbolException;
}
```

Purpose: Defines the abstraction for Braille conversion

Design Pattern: Strategy Pattern - allows interchangeable conversion algorithms

BrailleMap Class

```
    @Override
    public String convert(char ch) throws InvalidSymbolException {
        ch = Character.toLowerCase(ch); // Case normalization
        if (!braille.containsKey(ch)) {
            throw new InvalidSymbolException("Invalid symbol: " + ch);
        }
        return braille.get(ch); // Return 6-bit Braille code
    }
}
```

```java
public class BrailleMap implements BrailleConverter {
    protected HashMap<Character, String> braille = new HashMap<>();

    public BrailleMap() {
        loadBraille(); // Initialize mappings
    }

    private void loadBraille() {
        // Populate HashMap with Braille codes
        // Letters a-z: 6-bit binary patterns
        // Numbers 0-9: Same as letters a-j with number sign
        // Special characters: Punctuation and symbols
    }
```

Key Features:

Lazy Initialization: Mappings loaded in constructor

Case Insensitivity: Converts to lowercase for lookup

Error Handling: Throws custom exception for invalid characters

TextToBraille Class

```java
public class TextToBraille extends BrailleMap {

   public String convertText(String text) throws InvalidSymbolException {
       StringBuilder r1 = new StringBuilder(); // Row 1: dots 1 & 4
       StringBuilder r2 = new StringBuilder(); // Row 2: dots 2 & 5
       StringBuilder r3 = new StringBuilder(); // Row 3: dots 3 & 6

       boolean numberMode = false; // Tracks if we're in number sequence

       for (char ch : text.toCharArray()) {
           // Handle uppercase letters
           if (Character.isUpperCase(ch)) {
               add("000001", r1, r2, r3); // Capital sign
           }
```

```
            // Handle numbers
            if (Character.isDigit(ch) && !numberMode) {
                add(braille.get('#'), r1, r2, r3); // Number sign
                numberMode = true;
            }

            // Reset number mode for non-digits
            if (!Character.isDigit(ch)) {
                numberMode = false;
            }

            // Convert character and add to rows
            add(convert(Character.toLowerCase(ch)), r1, r2, r3);
        }

        return r1 + "\n" + r2 + "\n" + r3; // Return formatted 3-row output
    }
```

```
    private void add(String p, StringBuilder r1,
                     StringBuilder r2, StringBuilder r3) {
        // Format: dot1 space dot4 three-spaces
        r1.append(p.charAt(0)).append(" ").append(p.charAt(3)).append("   ");
        r2.append(p.charAt(1)).append(" ").append(p.charAt(4)).append("   ");
        r3.append(p.charAt(2)).append(" ").append(p.charAt(5)).append("   ");
    }
}
```

Algorithm Logic:

Capital Letters: Prefixed with 000001 (Braille capital sign)

Numbers: Prefixed with 001111 (Braille number sign) at start of sequence

Formatting: Each Braille cell displays as two columns with spacing

VisionDotsFX Class

```
public class VisionDotsFX extends Application {

    @Override
    public void start(Stage stage) {
        // UI Components
        Label title = new Label("VISION DOTS");
        TextArea input = new TextArea();
        TextArea output = new TextArea();
        Button convert = new Button("Convert to Braille");

        // Event Handler
        convert.setOnAction(e -> {
            try {
                TextToBraille tb = new TextToBraille();
                output.setText(tb.convertText(input.getText()));
            } catch (Exception ex) {
                output.setText("Error: " + ex.getMessage());
            }
        });
```

```
        // Layout and Styling
        VBox layout = new VBox(15, title, subtitle, input, convert, output);
        layout.setPadding(new Insets(20));

        // Scene Setup
        stage.setScene(new Scene(layout, 720, 540));
        stage.show();
    }
}
```

Braille Encoding Scheme

Letter Mapping Examples:

a = 100000 (● ∘ ∘ ∘ ∘ ∘)

b = 110000 (● ● ∘ ∘ ∘ ∘)

c = 100100 (● ∘ ∘ ● ∘ ∘)

Number Encoding (with number sign prefix):

1 = 001111 100000

2 = 001111 110000

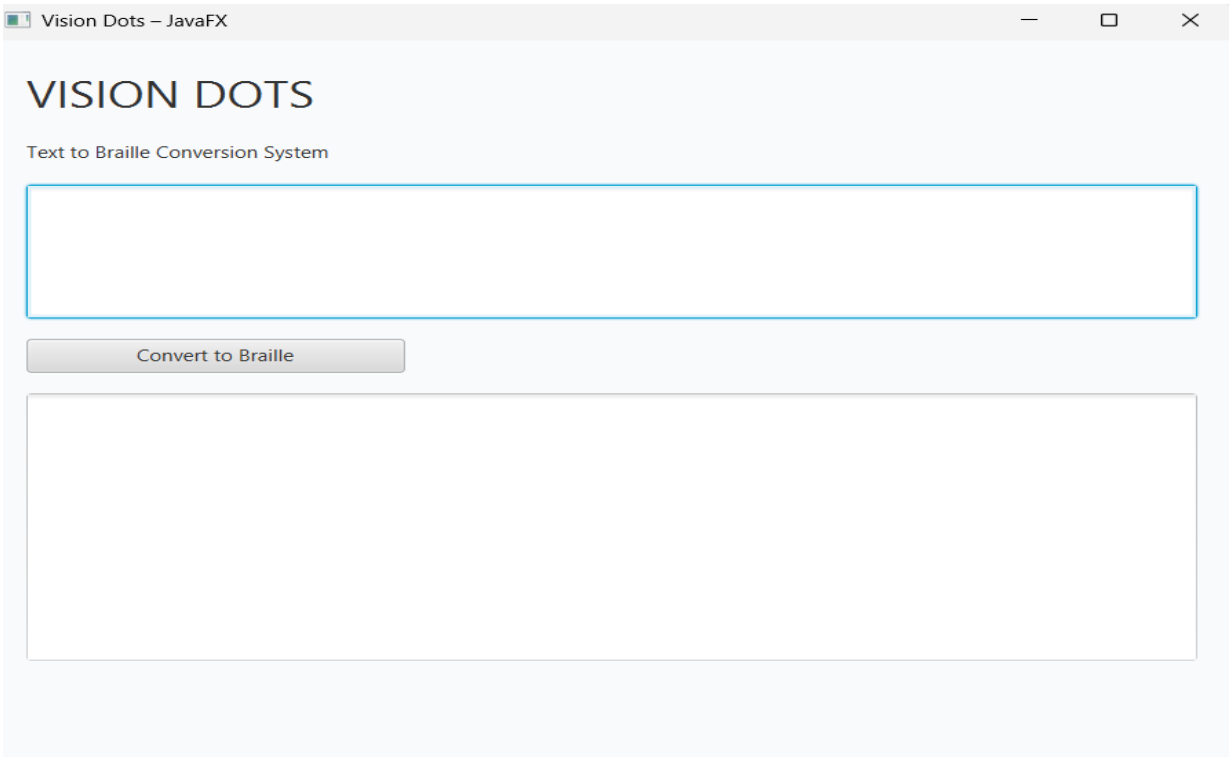
Special Signs:

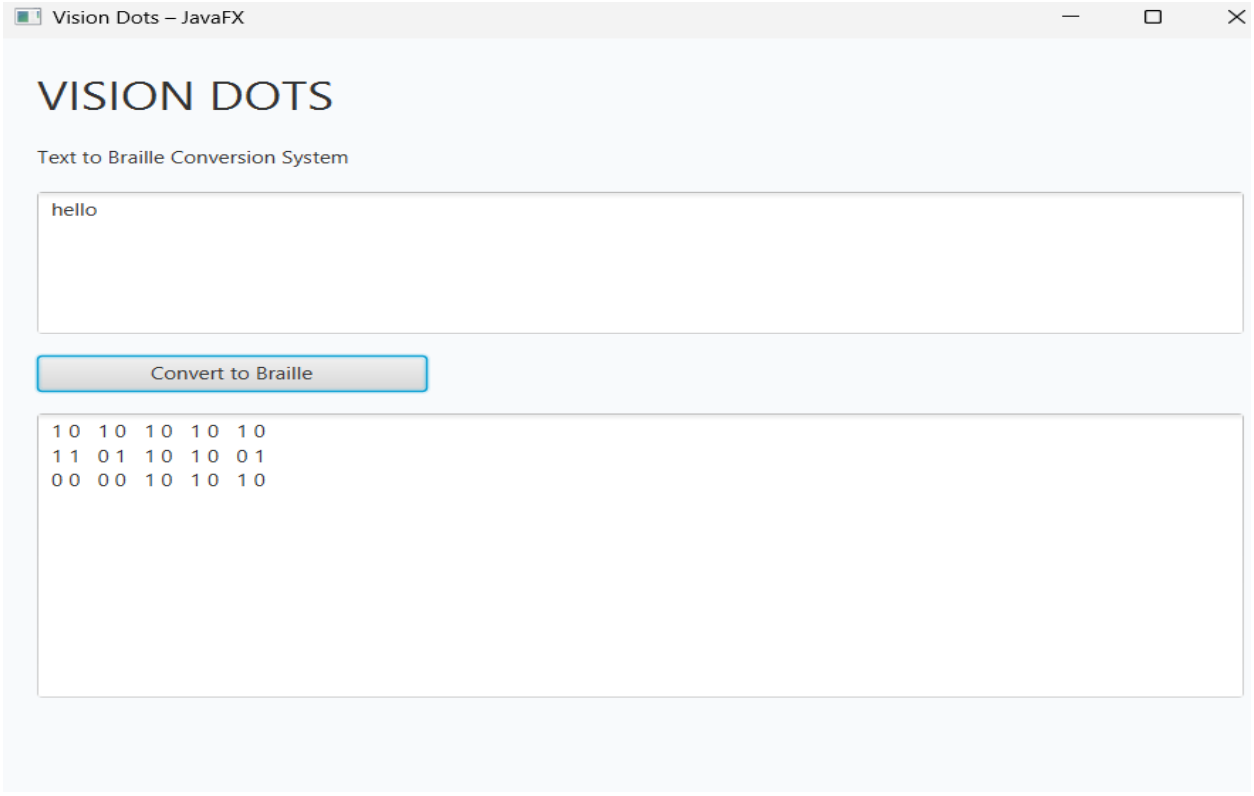
Capital: 000001

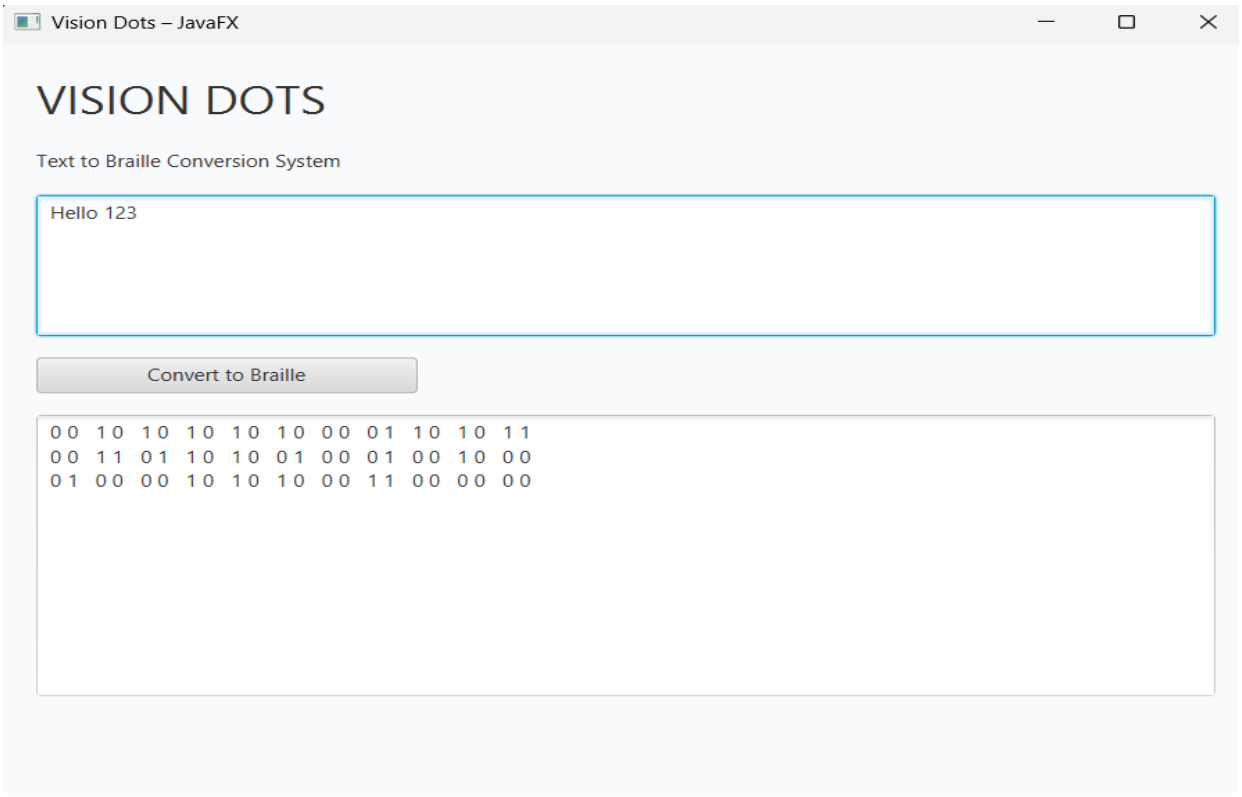
Number: 001111

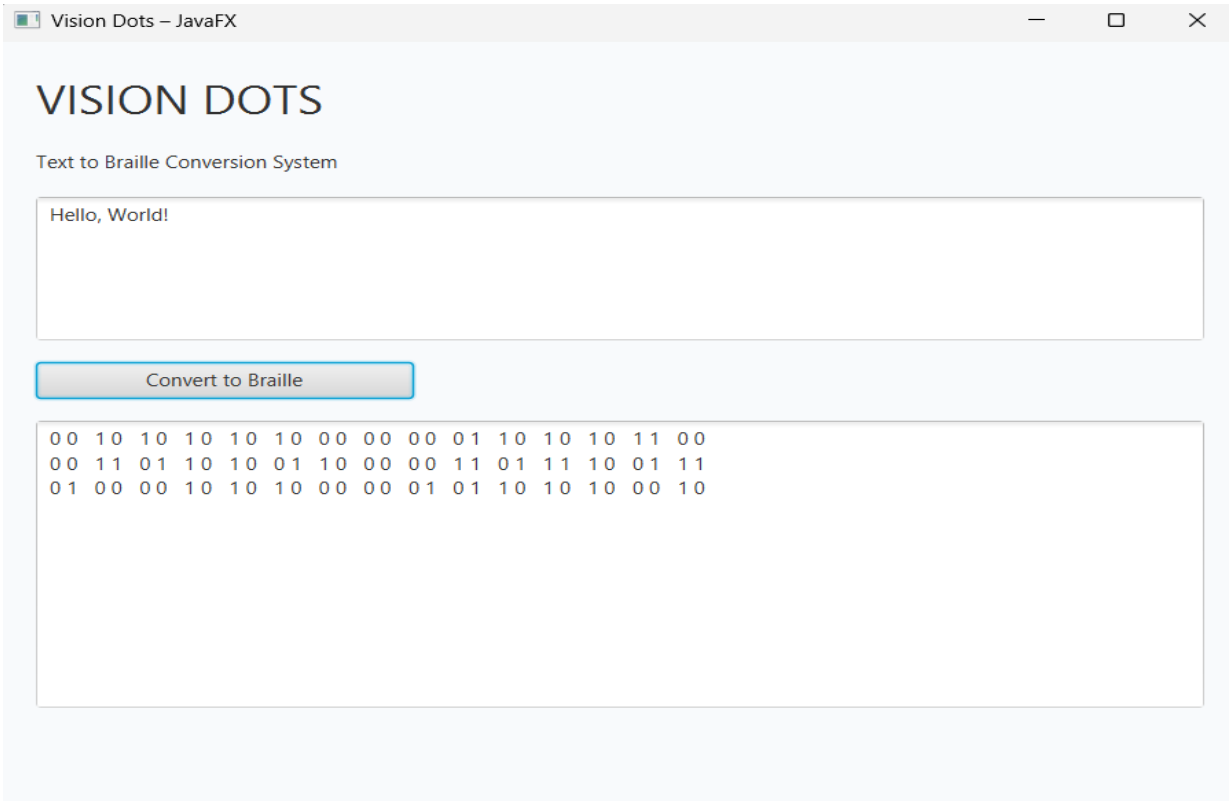
Space: 000000

Result


Vision Dots – JavaFX
VISION DOTS
Text to Braille Conversion System
Convert to Braille


Vision Dots – JavaFX
VISION DOTS
Text to Braille Conversion System
hello
Convert to Braille
10 10 10 10 10
11 01 10 10 01
00 00 10 10 10


Vision Dots – JavaFX
VISION DOTS
Text to Braille Conversion System
Hello 123
Convert to Braille
00 10 10 10 10 10 00 01 10 10 11
00 11 01 10 10 01 00 01 00 10 00
01 00 00 10 10 10 00 11 00 00 00


Vision Dots – JavaFX
VISION DOTS
Text to Braille Conversion System
Hello, World!
Convert to Braille
00 10 10 10 10 10 00 00 00 01 10 10 10 11 00
00 11 01 10 10 01 10 00 00 11 01 11 10 01 11
01 00 00 10 10 10 00 00 01 01 10 10 10 00 10

Conclusion

Project Achievements

Successfully implemented a complete Braille conversion system

Applied core OOP principles effectively

Created a graphical user interface

Handled edge cases and exceptions properly

Demonstrated educational value in Braille learning

Technical Strengths

Modular Design: Easy to maintain and extend

Scalability: Can add new characters/symbols easily

User-Friendly: Clean GUI with error handling

Educational: Clear visualization of Braille patterns

Current Limitations:

Supports only English characters

No support for contracted Braille (Grade 2)

No audio or tactile output

Future Enhancements:

Support for multiple languages

Grade 2 Braille contraction implementation

Audio feedback for visually impaired users

Export functionality (PDF, text file)

Mobile application version

References

Official Documentation

Oracle Java Documentation: https://docs.oracle.com/javase/

JavaFX Documentation: https://openjfx.io/

Unicode Braille Patterns: https://unicode.org/charts/PDF/U2800.pdf

Technical Resources

Braille Authority of North America. (2020). Braille Codes and Formats

World Blind Union. (2019). International Guide to Braille Notation

Java Design Patterns: https://java-design-patterns.com/

Academic References

Deitel, P. J., & Deitel, H. M. (2017). Java: How to Program

Bloch, J. (2018). Effective Java

Horstmann, C. S. (2019). Core Java Volume I: Fundamentals

Online Resources

W3Schools Java Tutorial: Basic Java concepts