“REUSABLE CAPTCHA”

A Project Report Submitted

For

Bachelor of Technology

In

INFORMATION TECHNOLOGY

By

HARSH SHARMA(2401330130120)
PRIYANSHU YADAV(2401330130198)
PUNIT SHARMA(2401330130199)

Under the Supervision of
Ms.Neetu Kumari Rajput
Assistant Professor,CSE Department

Department of Information Technology
School of Computer Science & Information Technology
NOIDA INSTITUTE OF ENGINEERING AND TECHNOLOGY, GREATER NOIDA
(An Autonomous Institute)
Affiliated to
DR. A.P.J. ABDUL KALAM TECHNICAL UNIVERSITY, LUCKNOW
November, 2025

DECLARATION

We hereby declare that the work presented in this report entitled “**REUSABLE CAPTCHA**”, was carried out by us. We have not submitted the matter embodied in this report for any other University or Institute. We have given due credit to the original authors/sources for all the words, ideas, diagrams, graphics, computer programs, experiments, results, that are not my original contribution. We have used quotation marks to identify verbatim sentences and given credit to the original authors/sources.

Name : HARSH SHARMA
Roll Number :2401330130120

(Candidate Signature)

Name :PRIYANSHU YADAV
Roll Number :2401330130198

(Candidate Signature)

Name :PUNIT SHARMA
Roll Number :2401330130199

(Candidate Signature)

CERTIFICATE

Certified that **HARSH SHARMA**(2401330130120), **PRIYANSHU YADAV**(2401330130198), **PUNIT SHARMA**(2401330130199) have carried out the mini project work presented in this report entitled “**REUSABLE CAPTCHA”** for the **Bachelor of Technology**, **Information Technology** from Dr. APJ Abdul Kalam Technical University, Lucknow_under our supervision. The Project Report embodies results of original work, and studies are carried out by the students herself/himself.

Signature	Signature
(Ms.Neetu Rajput)	(Dr Ritesh Rastogi)
(Asst. Professor)	Professor (HOD,IT)
Information Technology	Information Technology
NIET Greater Noida	NIET Greater Noida

Date:

ACKNOWLEDGEMENT

We would like to express my gratitude towards **Ms.Neetu Kumari Rajput** for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

Our thanks and appreciation to respected Dr Ritesh Rastogi, Dr Prabha S Nair, for their motivation and support throughout.

ABSTRACT

The Reusable CAPTCHA System developed in this project is a lightweight, standalone Java-based security module designed to distinguish human users from automated scripts using randomly generated text CAPTCHA strings. CAPTCHA mechanisms act as the first line of defense against automated attacks, protecting applications from bot-driven misuse, unauthorized access attempts, and automated form submissions. This project implements a six-character lowercase text CAPTCHA using Java Swing, demonstrating principles of event-driven programming, modular design, and user-interface development.

The system generates a new CAPTCHA using Java's randomness utilities and presents it to the user through an interactive GUI built with Swing components such as JFrame, JLabel, JButton, and JTextField. Users can refresh the CAPTCHA for better readability or verify their input, receiving instant feedback via dialog messages. Since the system operates entirely offline and performs all validation locally, it is highly suitable for academic demonstrations, classroom projects, and low-risk desktop applications.

A key design objective of the project is reusability. The CAPTCHA generator and validator are modular, enabling seamless integration into larger authentication workflows. Although contemporary CAPTCHA systems rely on advanced image-based or AI-based challenges, text-based CAPTCHAs remain relevant for low-security and offline environments due to their simplicity and minimal resource requirements. The interface follows usability principles by ensuring clear text, readable spacing, and interactive controls that enhance user experience.

Overall, this project demonstrates essential GUI programming concepts in Java, the use of randomness for security logic, and human-computer interaction considerations. The Reusable CAPTCHA module offers a clean, functional, and educational example of how fundamental verification systems can be designed, implemented, and tested using core Java technologies.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) systems serve as a fundamental security mechanism designed to differentiate human users from automated programs or bots. They play a crucial role in preventing unauthorized automated activities such as mass registrations, spam submissions, brute-force login attempts, and malicious script-based interactions across digital platforms . Despite advancements in artificial intelligence and modern CAPTCHA technologies, simple text-based CAPTCHAs continue to be widely used, especially in low-risk, offline, or academic environments due to their minimal computational requirements and ease of implementation .

The Reusable CAPTCHA System developed in this project is a lightweight Java-based application created using Java Swing components[1]. The system generates a six-character lowercase CAPTCHA string using Java's Random class and displays it through a clean and interactive graphical interface. Users can refresh the CAPTCHA or verify their input, with instant feedback provided through event-driven validation logic . This simple yet effective system demonstrates key concepts of GUI development, randomness generation, and client-side input validation, making it particularly well-suited for educational and demonstrative purposes .

The interface utilizes Swing elements such as JFrame, JLabel, JTextField, and JButton, ensuring a user-friendly experience supported by clear typography and consistent visual alignment. The refresh feature enhances usability by allowing users to request a clearer CAPTCHA whenever characters appear confusing or ambiguous. The verification feature compares the entered text directly with the internally stored CAPTCHA string, showcasing a foundational model of local authentication mechanisms .

A major strength of the system lies in its modularity and reusability. The CAPTCHA generation logic is designed to function as a standalone component, making it suitable for integration into a variety of applications such as login portals, desktop tools, or form-based software systems . Its offline nature ensures complete operability without the need for server-side processing or external APIs, increasing its portability and suitability for low-resource environments .

In summary, the Reusable CAPTCHA System provides an efficient demonstration of Java-based GUI programming, randomness-driven security logic, and human–computer interaction principles. Through its simplicity, clarity, and modular structure, the system offers an academically valuable example of how basic verification mechanisms can be developed and integrated into larger authentication frameworks.

1.1 Background of the Problem:

CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) systems are widely used as a basic security layer to differentiate real human users from automated scripts or bots. They help protect online and offline applications from malicious activities such as automated login attempts, mass registrations, spam submissions, and unauthorized form interactions . While many modern CAPTCHA implementations rely on image recognition, audio patterns, or puzzle-based logic, text-based CAPTCHAs continue to remain relevant due to their simplicity, accessibility, and low computational requirements in low-risk environments .

The Reusable CAPTCHA system developed in this project is a lightweight Java application built using Java Swing components. It generates random lowercase text strings which the user must correctly retype to verify human identity. The interface includes the CAPTCHA display, a user input field, and options to refresh or validate the generated CAPTCHA. This small-scale security component is especially suitable for academic purposes, offline applications, classroom demonstrations, or basic desktop software requiring elementary verification without server-based dependencies .

The system demonstrates essential concepts of GUI programming, randomness generation, input validation, and event-driven interaction in Java. By using Swing elements such as JFrame, JLabel, JButton, and JTextField, the application provides a clear, intuitive environment in which users can interact with the CAPTCHA . The refreshing mechanism allows users to request a new CAPTCHA, improving usability and accommodating cases where characters might appear confusing or ambiguous. The verification logic compares the user input with the generated CAPTCHA string, reinforcing client-side validation principles .

Additionally, the project emphasizes reusability — the CAPTCHA generator can be embedded into other Java applications, integrated into login forms, or adapted for more advanced verification systems . Due to its modular design and minimal resource usage, the application serves as a practical model for understanding the foundational logic behind CAPTCHA systems and for exploring the interaction between human users and simple automated security challenges .

1.1.1Problem Statement

Automated bots increasingly threaten digital systems by performing unauthorized activities such as spam submissions, repeated login attempts, and automated form interactions. While modern CAPTCHA solutions employ complex image- or audio-based mechanisms, these approaches often require significant computational resources, internet connectivity, or server-side processing, making them unsuitable for offline or academic environments . Additionally, sophisticated CAPTCHAs may introduce usability issues, causing difficulty for users due to ambiguous characters, accessibility barriers, or excessive visual distortion .

There is a clear need for a simple, lightweight, and reusable CAPTCHA mechanism that can operate entirely offline while ensuring basic protection against automated inputs. Existing solutions frequently lack modularity, making them hard to embed into different applications without major modifications . Moreover, many text-based CAPTCHA generators fail to maintain consistent randomness, reducing their effectiveness against pattern recognition and automated solving attempts .

Therefore, the problem addressed in this project is:

To design and develop a reusable, Java-based text CAPTCHA system that generates random, human-readable CAPTCHA strings, validates user input locally, operates without server dependency, and provides a clear, accessible interface suitable for academic, offline, and low-security applications .

This system aims to address the limitations of existing CAPTCHA mechanisms by focusing on modularity, usability, readability, and complete offline functionality while demonstrating core concepts of Java Swing, event-driven programming, and randomness-based security .

1.2 Identified Issues / Research Gaps

Existing CAPTCHA technologies present several limitations:

1. Traditional text CAPTCHAs are increasingly vulnerable to modern OCR and AI-based attacks, reducing their effectiveness in high-risk environments .
2. Complex visual CAPTCHAs often compromise usability, leading to user frustration and increased error rates .
3. Image and audio CAPTCHAs may not be accessible to visually impaired or differently-abled users .
4. Many CAPTCHA systems depend on server-side engines or external APIs, limiting their use in fully offline environments.
5. Numerous implementations are tightly integrated with parent applications, reducing modularity and reusability .
6. Frequent misrecognition occurs due to ambiguity between similar characters, affecting human–computer interaction .

These issues demonstrate a strong need for a CAPTCHA system that is simple, readable, offline-capable, and reusable, especially for academic or small-scale environments.

1.3 Objectives and Scope

Objectives

The main objectives of the Reusable CAPTCHA project are:

- To develop a desktop-based CAPTCHA system using Java Swing, enabling users to view, refresh, and verify text-based CAPTCHAs in real time .
- To apply randomness principles using Java's `Random` class, ensuring unpredictable character selection .
- To build a reusable and modular CAPTCHA component suitable for embedding into authentication forms or standalone software .
- To enhance user interaction through a clean and accessible graphical interface with minimal complexity .
- To demonstrate client-side verification logic by comparing user input with the generated CAPTCHA string .

Scope

The scope of the project includes:

- Generating six-letter lowercase CAPTCHA strings .
- Allowing user input and providing verification with instant feedback .
- Offering refresh functionality for better readability .
- Ensuring a simple, GUI-based front-end using Java Swing .
- Supporting complete offline operation without servers .

Out of Scope:

- No database integration
- No image or audio-based CAPTCHA
- No encryption or advanced security algorithms
- No web-based deployment or backend support

These limitations help maintain focus on core academic and educational goals.

1.4 Project Report Organization

This project report is organized into the following chapters:

- Chapter 1: Introduction — Background, research gaps, objectives, and scope
- Chapter 2: Literature Review — Overview of existing CAPTCHA systems and limitations
- Chapter 3: Requirements — Functional, non-functional, hardware, and software requirements
- Chapter 4: System Design — Architecture, GUI design, and implementation details
- Chapter 5: Testing & Results — Methodology, test cases, and performance evaluation
- Chapter 6: Conclusion — Summary of findings
- Chapter 7: Future Work — Recommendations for system enhancement .

CHAPTER 2
LITERATURE SURVEY

2.1 Existing Systems and Tools

CAPTCHA systems have evolved significantly since their introduction, originating as simple mechanisms intended to differentiate real human users from automated programs. The earliest CAPTCHA models primarily relied on distorted text-based challenges, where users were required to identify and retype characters from randomized sequences . These early systems were successful because traditional bots lacked the advanced optical recognition capability required to decode such images. As a result, text-based CAPTCHAs quickly became a standard security practice used by websites, online forms, and applications seeking protection against automated misuse.

However, the rapid development of artificial intelligence—specifically machine learning and OCR technologies—reduced the effectiveness of classic text CAPTCHAs. Modern OCR engines are capable of recognizing patterns with high accuracy, even when distortion techniques are applied . To counter this, researchers introduced alternative CAPTCHA approaches, such as image classification tasks, puzzle-solving tests, and audio-based challenges. Image-based CAPTCHAs require users to identify specific objects, scenes, or categories from a collection of images. While more secure, these systems introduce additional computational cost and require stable network connectivity, making them less suitable for offline applications .

Audio CAPTCHAs were introduced as an accessibility measure for visually impaired users, but they too present several issues. Background noise, unclear pronunciation, accent variations, and audio distortion can make solving such CAPTCHAs difficult even for human users . Puzzle-based CAPTCHAs, such as drag-and-drop challenges or logic-based tasks, offer high security but increase cognitive load and are often unsuitable for general-purpose applications.

In contrast, lightweight CAPTCHA solutions designed for desktop-based or offline systems continue to rely on simplicity and readability. Java-based CAPTCHA modules—such as the one developed in this project—focus on generating random text strings, validating user input, and providing a clear interface suitable for academic and small-scale software development environments. Their simplicity, modularity, and low processing requirements make them ideal for demonstrating the fundamentals of human–computer interaction, randomness generation, and event-driven programming .

2.2 Comparative Study

The following observations highlight the key differences across commonly used CAPTCHA techniques:

System Type	**Features**	**Advantages**	**Limitations**
Traditional Text CAPTCHA	Random letters & digits	Simple, fast, easy to implement	Vulnerable to OCR attacks
Image/Visual CAPTCHA	Select images of objects	Difficult for bots	High user difficulty, accessibility issues
Audio CAPTCHA	Spoken digits/words	Supports visually impaired users	Noisy audio, difficult accents
Puzzle CAPTCHA	Drag-drop or logic tasks	Interactive, bot-resistant	High complexity, not suitable for offline use

Compared to these systems, the **Reusable CAPTCHA System** developed in this project prioritizes:

- **Offline operation** (no dependence on external APIs)
- **Simplicity and readability** (lower risk of user error)
- **Modularity** (easy integration into other applications)
- **Low resource usage** (ideal for academic environments)

2.2.1 Comparative Evaluation Based on Security, Usability, and Accessibility

To further evaluate CAPTCHA systems in a detailed academic context, it is essential to examine them across three key criteria: **security**, **usability**, and **accessibility**. Security refers to the system's ability to resist automated attacks, usability focuses on the user's ease of interaction, and accessibility ensures that the system can be used by individuals with disabilities.

From a security perspective, **text-based CAPTCHAs** provide low to moderate protection. While early models were highly effective, advancements in OCR mean that many bots can now decode common text patterns with high accuracy . **Image-based CAPTCHAs** are comparatively stronger, as they rely on complex object recognition tasks that remain challenging for many automated systems. **Audio CAPTCHAs**, although originally introduced to improve accessibility, often suffer from clarity issues and can be vulnerable to speech-to-text tools after noise reduction processing .

Usability is equally important. Simple text CAPTCHAs score highest on user friendliness due to their quick interaction time and familiar structure. In contrast, image-based CAPTCHAs often require multiple attempts because users struggle with ambiguous images or unclear object boundaries . Puzzle-based CAPTCHAs offer interactive engagement but significantly slow down user workflow, making them unsuitable for high-traffic platforms .

Accessibility remains a major challenge for most advanced CAPTCHA types. Text CAPTCHAs may not be suitable for individuals with visual impairments unless supported by audio alternatives. Image CAPTCHAs are particularly problematic for low-vision users, while audio CAPTCHAs can be incomprehensible for users with hearing difficulties or non-native language backgrounds .

The **Reusable CAPTCHA System** designed in this project positions itself as a high-usability, moderate-security solution that prioritizes readability and simplicity over advanced protection. It is best suited for offline, academic, and low-risk environments where ease of integration and minimal computational requirements are essential .

2.3 Limitations of Existing Systems

Despite their effectiveness, existing CAPTCHA technologies present several limitations across academic, desktop, and commercial implementations.

1. Vulnerability to OCR and AI-Based Attacks

As machine learning models improve, traditional text CAPTCHAs become increasingly susceptible to automated solving. Even distorted text CAPTCHAs are often cracked by modern OCR algorithms, reducing their security effectiveness in high-risk environments . This makes text-based CAPTCHAs suitable primarily for low-security applications.

2. Usability Issues in Complex CAPTCHAs

Many modern CAPTCHA implementations, especially image- or pattern-based ones, compromise usability. Users often struggle with “select all squares with traffic lights” or “identify crosswalks,” leading to frustration and increased login or form abandonment rates .

In contrast, text CAPTCHAs—while simpler—may still suffer from ambiguity when certain characters look similar.

3. Accessibility Concerns

Sophisticated image-based CAPTCHAs may not be accessible to visually impaired users or those relying on screen readers. Audio-based CAPTCHAs attempt to address this issue but often introduce their own usability problems due to background noise, accent variation, or low audio quality . Text CAPTCHAs remain more accessible but still require clear font choices and appropriate visual design.

4. Dependence on Web-Based Infrastructure

Many CAPTCHA solutions rely on server-side validation, cloud-hosted engines, or external APIs. This dependency poses challenges in environments with limited or no internet connectivity. Offline CAPTCHA solutions—such as the Reusable CAPTCHA system—avoid these limitations but lack the advanced adaptive security of online systems .

5. Overhead of Maintaining Large CAPTCHA Databases

Graphical CAPTCHA systems often require extensive labeled image datasets and regular updates to remain secure against AI attacks. Maintaining such datasets increases development overhead and reduces portability, making them unsuitable for academic and small prototype systems .

6. Limited Reusability in Traditional Implementations

Some CAPTCHA systems are tightly coupled with the applications they secure, reducing opportunities for modular reuse. Reusable CAPTCHA components, such as the one developed in this project, solve this problem by providing a modular, standalone CAPTCHA generator that can be embedded in various interfaces and applications .

7. Ambiguity in Character Recognition

Even simple text CAPTCHAs suffer from confusion between similar characters, such as 'l' and '1' or 'o' and '0'. This reduces usability and increases error rates, affecting overall human–computer interaction efficiency .

8. Lack of Personalization or Adaptability

Most CAPTCHA systems do not adapt to user behavior or threat patterns. Adaptive CAPTCHAs exist but require backend intelligence, machine learning models, and analytics engines. Lightweight CAPTCHA systems remain static, limiting their ability to respond to emerging automated threats .

9. Potential for Automated Bypass Tools

Free and commercial tools exist that specifically target breaking common CAPTCHA formats. Simple CAPTCHAs are often bypassed using scripting frameworks, browser automation tools, or pre-trained solver models, making them unsuitable for scenarios requiring high security .

2.4 Need for the Proposed System

The Reusable CAPTCHA System addresses the shortcomings of existing systems by focusing on:

- Complete offline functionality with no server dependency.
- Simple, lowercase alphabetic CAPTCHAs that avoid ambiguous characters.
- High readability and user-friendly interface following usability research .
- Modular architecture that separates GUI, logic, and event handling layers for easy integration .
- Lightweight performance suitable even for low-end machines or classroom environments .

The increasing sophistication of automated bots has made CAPTCHA systems indispensable. However, the majority of modern CAPTCHAs rely on server-side validation, cloud APIs, or high-complexity interactions. This creates accessibility, usability, and deployment challenges for environments that require lightweight or offline-ready systems. Academic institutions, local desktop applications, student projects, and prototyping environments frequently require CAPTCHA solutions that are simple, efficient, and easy to integrate into various software modules .

The proposed Reusable CAPTCHA System addresses this need by focusing on offline operability and modularity. Because it is implemented entirely using Java Swing and runs locally without network dependencies, it can be integrated into any desktop-based application without requiring backend servers or external services . This makes it ideal for low-risk applications where simplicity and portability are more important than high-security authentication. Thus, the system fulfills a specific niche: providing an educational, accessible, and reusable CAPTCHA that demonstrates core programming concepts without the complexities of modern web-based verification systems.

CHAPTER 3
SYSTEM REQUIREMENTS AND ANALYSIS

3.1 Introduction

This chapter presents a comprehensive analysis of the requirements necessary for developing and deploying the Reusable CAPTCHA System. A well-defined requirement analysis ensures that the project meets the intended objectives, operates efficiently within defined constraints, and delivers a usable and functional solution. Requirement analysis helps identify the resources, system behavior, performance expectations, and software interactions needed for successful development. As CAPTCHA systems serve a critical role in distinguishing human interaction from automated scripts, defining these requirements becomes essential for ensuring reliability and usability .

The Reusable CAPTCHA System is a lightweight Java-based application designed for offline and academic environments. Its system requirements differ significantly from complex online CAPTCHA frameworks that depend on large databases, image processing libraries, or server-side APIs. Instead, the proposed system requires minimal hardware resources, making it suitable even for low-end machines or student-level environments where high-performance computing may not be available . The Java Swing framework enables platform independence, ensuring that the application can operate on any system supporting Java Runtime Environment (JRE).

The software requirements include the Java Development Kit (JDK), Java Swing GUI libraries, and optional development tools like NetBeans or IntelliJ IDEA. Since the system is designed to run offline, it does not require network connectivity, external storage, or backend servers. This lowers deployment complexity and supports portability across different environments .

Additionally, analyzing functional and non-functional requirements ensures that the system behaves predictably under various user scenarios. Functional requirements describe what the system should do—such as generating CAPTCHAs, validating input, or refreshing the interface. Non-functional requirements describe system quality attributes like performance, security level, usability, reliability, and accessibility. These elements ensure that the application not only works as expected but also provides a smooth user experience .

Thus, this chapter establishes the foundation upon which the rest of the system design and implementation is built, ensuring clarity, structure, and consistency in the development process.

3.2 Requirements Specification

3.1.1 Functional Requirements

Functional requirements describe the essential operations the Reusable CAPTCHA system must perform to achieve its intended purpose. These requirements ensure that the system behaves correctly and provides a seamless verification process for users across different interactions .

1. CAPTCHA Generation

The system must generate a random six-letter lowercase CAPTCHA string using Java's Random class. Each character must be selected unpredictably from the alphabet to prevent pattern recognition by automated tools .

2. CAPTCHA Display

A readable, visually clear CAPTCHA must be displayed within the GUI using a JLabel. The interface must preserve consistent font style, spacing, and visibility to enhance user comprehension and reduce recognition errors .

3. CAPTCHA Refreshing

The system must provide a "Refresh" button that replaces the current CAPTCHA with a newly generated string. This ensures accessibility and user comfort, allowing users to request a clearer CAPTCHA if the current one appears confusing .

4. User Input Field

A JTextField must be provided for users to manually type the CAPTCHA. The input field must accept alphanumeric input and allow editing before verification.

5. CAPTCHA Verification

The system must validate the user's input by comparing it directly with the generated CAPTCHA string. Verification must be case-sensitive to maintain accuracy and security .

6. Feedback Messages

If the input matches the CAPTCHA correctly, the system must display a success message. Otherwise, the system must notify the user of an incorrect attempt and prompt them to try again .

7. Error Handling

The system must handle invalid inputs such as empty fields, whitespace-only entries, or incorrect characters. Relevant alert dialogs must guide the user toward correct usage.

8. GUI Responsiveness

All user interactions—refreshing, entering input, and verification—must provide immediate visual and functional feedback to ensure a smooth user experience .

3.1.2 Non-Functional Requirements

Non-functional requirements describe the quality attributes, performance constraints, and design expectations necessary for the system's long-term stability and usability .

1. Usability

The interface must follow clean and simple design principles using Java Swing components. Fonts must be legible, spacing must be balanced, and user actions must require minimal steps .

2. Performance

The CAPTCHA must be generated instantly, and verification must occur without noticeable delay. The program must respond efficiently even after repeated refresh or verification actions.

3. Reliability

The system must operate without crashes or inconsistencies. CAPTCHA generation must always produce valid six-character strings, and verification must remain accurate under all conditions .

4. Security

The CAPTCHA must exhibit randomness in character selection to reduce predictability and resist automated guessing attempts. Although simple, it must prevent direct programmatic bypass without manual input .

5. Portability

The application must run on any system supporting Java 8 or above, including Windows, macOS, and Linux. No platform-specific dependencies should be required .

6. Maintainability

The code must follow modular Java structure to allow easy extension or integration into larger applications. The CAPTCHA generation logic must remain encapsulated for reuse .

7. Accessibility

The GUI must use clear fonts, proper spacing, and consistent lowercase characters to reduce confusion. Users must be able to refresh the CAPTCHA if readability concerns arise .

8. Scalability

Although simple, the system design must support future expansion, such as adding alphanumeric characters, distortion effects, or time-based expiration mechanisms .

3.3 Software and Hardware Requirements

This section outlines the minimum and recommended specifications required to run the Reusable CAPTCHA application effectively.

3.2 Hardware Requirements

Category	Minimum Requirement	Recommended Requirement
Processor	Intel Pentium IV or equivalent	Intel Core i3 or above
RAM	1 GB	2 GB or above
Storage	200 MB free space	500 MB free space
Display	1024 × 768 resolution	1366 × 768 or higher
Input Devices	Keyboard, Mouse	Keyboard, Mouse
Operating System Support	Windows / Linux / macOS	Latest version of Windows / Linux

Given the lightweight computational workload, these requirements ensure smooth execution of the Java Swing interface, random CAPTCHA generation, and user input handling without resource strain. The system does not use graphics-intensive rendering, allowing compatibility with older hardware. As long as the computer supports Java Runtime Environment (JRE), the application will run efficiently .

Software Requirements:

Software Type	**Specification**
Operating System	Windows 7/8/10/11, Linux Ubuntu 16+, macOS
Programming Language	Java (JDK 8 or higher)
GUI Framework	Java Swing
Development Environment (Optional)	NetBeans / Eclipse / IntelliJ IDEA
Runtime Environment	Java Runtime Environment (JRE)
Text Editor	Notepad++, VS Code, or any standard editor

The proposed system's simplicity eliminates the need for web servers, database systems, or third-party libraries. The application runs independently without requiring internet connectivity or backend integration, making it ideal for offline environments or situations where online components may be restricted .

The reliance on Java technology ensures stability, cross-platform support, and long-term maintainability. The development environment is flexible enough to accommodate both beginners and advanced users, supporting experimentation, testing, and efficient debugging.

Thus, the software stack ensures reliability, portability, and minimal installation overhead—all essential characteristics for a reusable and educational CAPTCHA system .

3.3 Preliminary Product Description / Modules Details

The Reusable CAPTCHA System is designed as a lightweight, modular Java application that provides a text-based CAPTCHA generation and validation mechanism suitable for academic, offline, and low-security environments. This system is composed of several interconnected modules that work together to ensure the generation of random CAPTCHA strings, display them through an intuitive graphical interface, and validate user input in real time. The modular architecture enhances maintainability, promotes reusability, and simplifies integration with other Java-based applications .

Module 1: CAPTCHA Generator Module

This module is responsible for generating a six-character random CAPTCHA string using Java's Random class. It uses lowercase alphabets and avoids ambiguous characters to enhance readability. This module forms the core logic of the entire system. It ensures randomness, unpredictability, and consistent refresh functionality, aligning with usability requirements .

Module 2: GUI Display Module (Java Swing Interface)

Implemented using Swing components such as JFrame, JLabel, JTextField, and JButton, this module handles the visual representation of the CAPTCHA. It arranges all interface elements using layout managers and ensures a user-friendly design following human–computer interaction principles. The GUI includes fields for displaying the CAPTCHA, receiving user input, and buttons for refreshing and validating the CAPTCHA .

Module 3: CAPTCHA Validation Module

This module verifies the user-entered text by comparing it with the generated CAPTCHA string. It displays appropriate messages such as "Correct CAPTCHA" or "Incorrect CAPTCHA" using dialog boxes. This validation process is handled completely offline, reinforcing the goal of creating a portable, independent verification system .

Module 4: Event Handling Module

This component manages event listeners associated with GUI elements. When the user clicks the refresh or validate button, the module triggers appropriate actions—either generating a new CAPTCHA or validating input. This module ensures smooth interaction between the logical and graphical components of the application .

Module 5: Reusability & Integration Module

The final module focuses on separating CAPTCHA logic into independent classes so that developers can integrate the CAPTCHA into login windows, registration forms, or standalone applications without rewriting the logic. This supports modularity and educational reuse.

Together, these modules form a structured and efficient Java-based CAPTCHA system that prioritizes simplicity, modularity, offline functionality, and educational value.

3.4 Use Case Diagram

The Use Case Diagram for the Reusable CAPTCHA System provides a high-level overview of how users interact with the system and how different system components respond to user actions. A use case diagram helps visualize the functional requirements by identifying the actors, their goals, and the system's interactions. In this project, the **primary actor** is the **End**

User, who interacts with the graphical interface to read the CAPTCHA, refresh it when required, and validate their input.

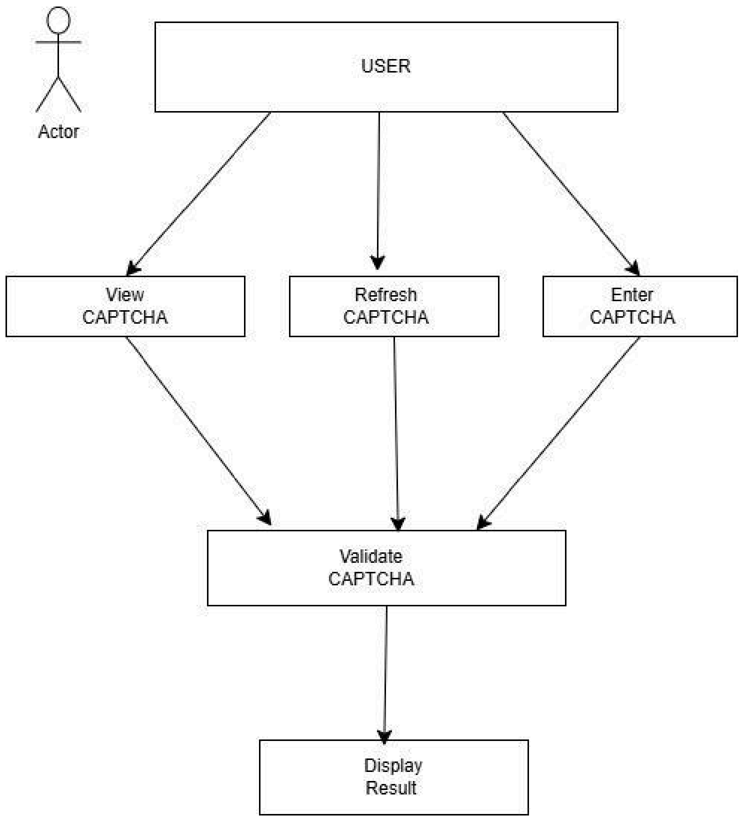
The system itself behaves as an automated component that generates CAPTCHA strings, shows them on the screen, and performs validation when triggered. The use case diagram includes several essential actions reflecting the operations in the CAPTCHA system .

Actors:

- **User:** The only external actor who interacts with the system interface.

Use Cases:

1. **View CAPTCHA:**
   The system displays a six-character CAPTCHA automatically when the application starts.
2. **Refresh CAPTCHA:**
   The user may request a new CAPTCHA if the current one appears unclear. Clicking the refresh button generates a new random string.
3. **Enter CAPTCHA Text:**
   The user types the CAPTCHA into the input field.
4. **Validate Input:**
   The user clicks the "Verify" button, triggering the system to compare the user input with the stored CAPTCHA string .
5. **Receive Result:**
   The system displays a success or failure message depending on the input accuracy .

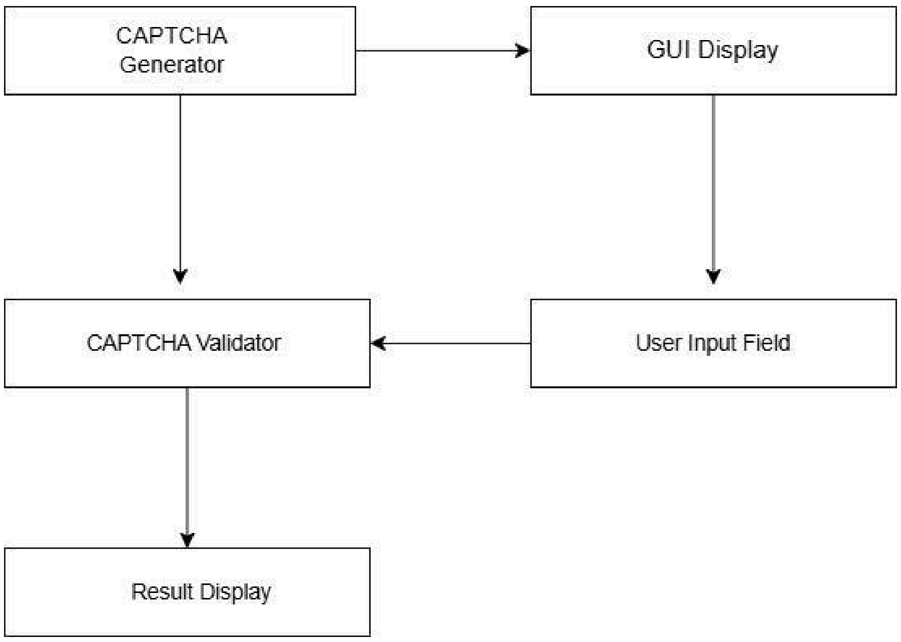


This diagram illustrates that the user initiates all interaction sequences. The system responds to each request by executing internal modules (generation, validation, GUI updates). There is no secondary actor since the CAPTCHA system is designed for offline standalone operation . Each use case corresponds directly to a module in the architecture, ensuring clear mapping between requirements and implementation. This structure also helps in identifying missing user flows, validating completeness, and simplifying future extensions—such as integrating audio CAPTCHA or character distortion.

CHAPTER 4
SYSTEM DESIGN

4.1 System Architecture

The architecture of the Reusable CAPTCHA System includes four primary components: the CAPTCHA Generator, GUI Interface, Validation Unit, and Event Handler. These components work together to generate random CAPTCHA strings, display them to users, accept user input, and validate the response .



- **CAPTCHA Generator:**
  Produces a random string of lowercase characters using Java's Random class. The generator ensures unpredictability while keeping characters readable .
- **GUI Display Module:**
  Built using Java Swing components (JFrame, JLabel, JButton, etc.). It shows the generated CAPTCHA and allows users to interact with the system .
- **Validation Module:**
  Compares the user-entered text with the actual CAPTCHA string. It triggers success or failure messages using dialog boxes .
- **Event Handling Module:**
  Listens for actions such as clicking the "Refresh" or "Verify" button. It ensures that GUI events correctly trigger underlying logic functions .

Overall, the architectural design provides clarity and ensures easy maintainability. Because the modules are loosely coupled, developers can modify one part—such as enhancing CAPTCHA complexity—without affecting the entire system.

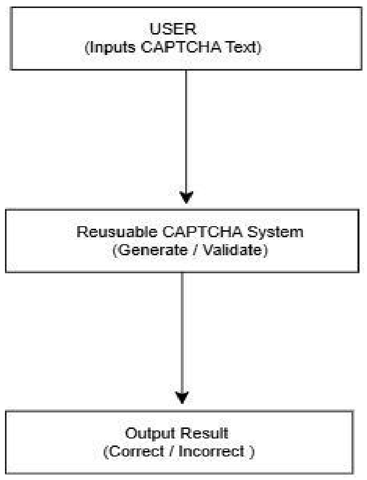
4.2 Data Flow Diagram

The Data Flow Diagram (DFD) provides a graphical representation of how data moves through the Reusable CAPTCHA System. It illustrates the flow of information between different components, external entities, and data stores. DFDs are widely used during the design phase because they simplify complex workflows and help developers understand the logical movement of data without focusing on technical implementation details . For this CAPTCHA system, DFDs demonstrate how input flows from the user, how the system processes the input, and how validation results are generated.

The Reusable CAPTCHA System is lightweight and operates entirely offline. Therefore, the DFD emphasizes simplicity and clarity, focusing only on the internal modules responsible for generating, displaying, and validating CAPTCHAs. There are no external databases or servers involved, which aligns with the project's aim to create a standalone, modular verification tool suitable for academic and desktop-based environments .

Below are the DFD models representing Level 0 and Level 1 diagrams.

4.2.1 DFD Level 0 (Context Diagram)

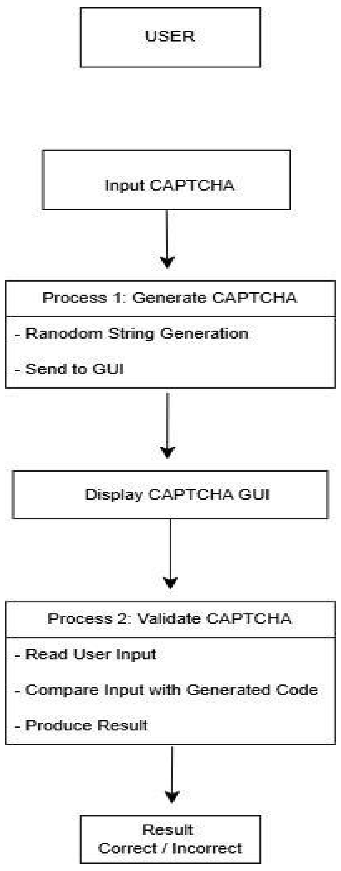


At Level 0, the system is treated as one process.

The user interacts with the system by viewing and entering CAPTCHA text.

The system processes this data and returns either a success or failure message .

4.2.2 DFD Level 1 (Detailed Flow)


USER
Input CAPTCHA
Process 1: Generate CAPTCHA
- Ranodom String Generation
- Send to GUI
Display CAPTCHA GUI
Process 2: Validate CAPTCHA
- Read User Input
- Compare Input with Generated Code
- Produce Result
Result
Correct / Incorrect

In Process 1: CAPTCHA Generation, the system creates a six-character random string using Java's built-in randomization utilities . This data is fed into the GUI module, which displays it for the user. There are no external data stores because the system operates entirely locally and temporarily stores the generated CAPTCHA in memory .

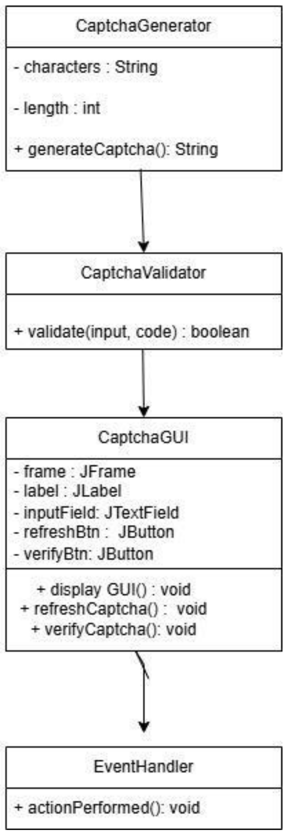
In Process 2: CAPTCHA Validation, the system collects input from the user through the GUI's text field. The validation module compares this input with the stored CAPTCHA string. If the values match, the system generates a success message; otherwise, it produces an error message .

This structured DFD layout highlights the clarity and efficiency of the Reusable CAPTCHA System's internal workflow, confirming that the entire process—from generation to validation—remains simple, modular, and user-friendly .

4. 3 UML Diagrams

4.3.1 Class Diagram

The class diagram represents the object-oriented structure of the system by showing classes, attributes, methods, and relationships. It visualizes how each component of the CAPTCHA system interacts internally, demonstrating the principles of modularity, encapsulation, and separation of concerns.



The **CaptchaGenerator** class is responsible for creating the random CAPTCHA string. It contains attributes such as the character set and CAPTCHA length. The **generateCaptcha()** method is the core functionality that produces a readable string .

The **CaptchaValidator** class performs comparison-based validation. It ensures the logic is independent from the GUI to improve reusability .

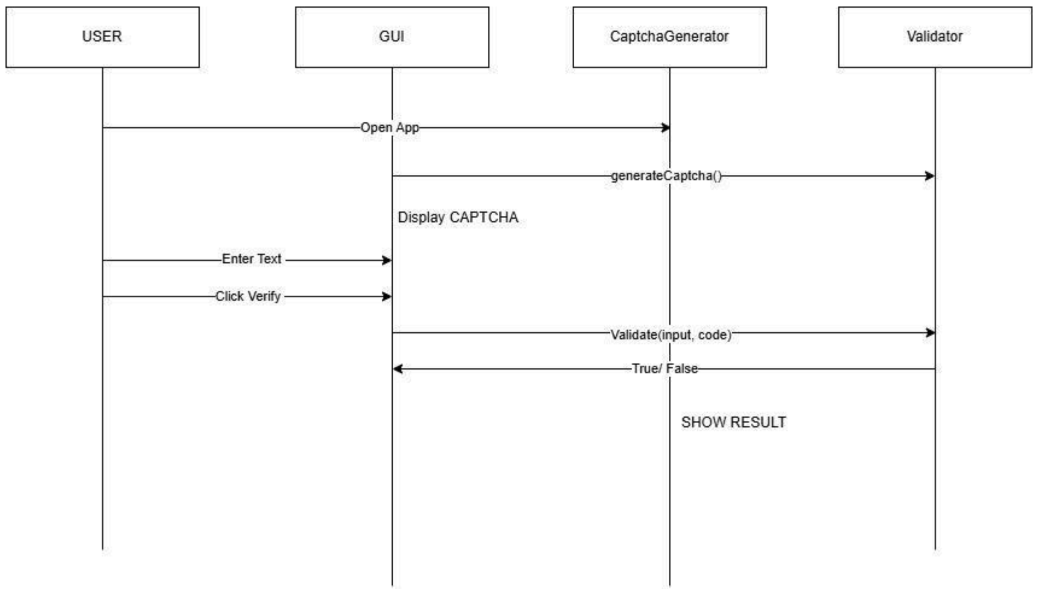
The **CaptchaGUI** class manages the user interface and integrates with the generator and validator classes. It contains all Swing components like buttons, labels, and text fields, creating a clean, user-friendly structure .

Finally, the **EventHandler** class manages button clicks and triggers validation or refresh processes .

This object-oriented design ensures that the system is scalable, modular, and easily integrable with other applications.

4.3.2 Sequence Diagram

The Sequence Diagram depicts the flow of interactions between system components during a complete user operation. It provides a detailed view of the order in which methods are executed when the user interacts with the CAPTCHA.



1. **Application Startup:**

   The GUI initializes and requests a CAPTCHA from the generator .
2. **CAPTCHA Generation:**

   The generator returns a random string, which is displayed on the GUI.
3. **User Interaction:**

   The user types the CAPTCHA and clicks “Verify.”
4. **Validation:**

   The input is passed to the Validator class.
   - If the input matches, the system sends a “success” result.
   - Otherwise, it returns “failure” .
5. **Result Display:**

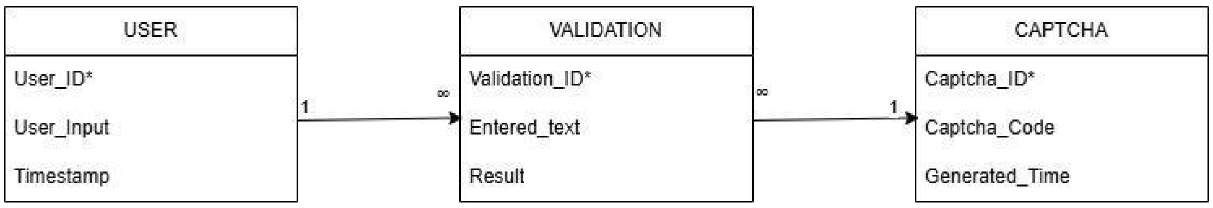
   The GUI presents feedback through dialog boxes .

This sequence visually represents the complete life cycle of CAPTCHA verification, depicting how each component interacts in a synchronized manner.

4.5 ER Diagram

Although the Reusable CAPTCHA System operates entirely offline without the need for external databases or permanent storage, an Entity–Relationship (ER) Diagram is still useful for academic documentation. It helps conceptualize the logical structure of the system by identifying the essential entities, their attributes, and the relationships between them. In this project, the ER diagram symbolizes how internal components interact at a conceptual data level, even though these interactions occur in memory rather than in a physical database .

The primary entities relevant to a CAPTCHA validation process include User, Captcha, and Validation. These entities represent key processes within the system, such as generating CAPTCHA strings, accepting user input, and validating responses. Each entity contains attributes used during execution—e.g., the CAPTCHA string, user-entered text, and validation result .



Explanation of Entities

1. CAPTCHA Entity

This entity stores the system-generated CAPTCHA code. Even though the system does not maintain a database, conceptually, the CAPTCHA entity holds:

- **Captcha_ID**
- **Captcha_Code**
- **Generated_Time**

This reflects the random string generation process, which uses Java's Random class to produce unpredictable and readable CAPTCHA text .

2. USER Entity

The User entity represents the individual interacting with the GUI. Its conceptual attributes include:

- **User_ID** (auto-generated session index)

- **User_Input**
- **Timestamp**

This representation aligns with usability and human–computer interaction principles .

3. VALIDATION Entity

This entity captures the validation process:

- **Entered_Text**
- **Result** (Correct / Incorrect)
- **Validation_ID**

The system compares the user’s input with the generated CAPTCHA string to produce the validation result .

Relationships

- **USER → VALIDATION (1:M):**
  A user can perform multiple validation attempts.
- **CAPTCHA → VALIDATION (1:M):**
  A single generated CAPTCHA may be validated multiple times until refreshed .
- **USER and CAPTCHA are indirectly connected** through validation attempts.

These relationships reflect how data flows logically during runtime, even without physical storage.

CHAPTER 5
METHODOLOGY & IMPLEMENTATION

5.1 Dataset Description

The Reusable CAPTCHA System does not use a conventional external dataset, as found in machine learning or database-driven applications. Instead, it relies on a synthetic dataset generated dynamically at runtime. This dataset consists of randomly generated CAPTCHA strings, formed using lowercase alphabetic characters (a–z). These strings act as both the security token and the data upon which validation is performed.

The internal dataset can be described as a character set and randomized sequences derived from it. The character set includes: abcdefghijklmnopqrstuvwxyz

Using this set, the system constructs a 6-character CAPTCHA string during each refresh cycle. This internally generated dataset ensures unpredictability and ensures that no two CAPTCHA strings are identical in sequence, thereby adding a basic layer of security .

Synthetic Dataset Characteristics:

1. **Dynamic Generation:**
   CAPTCHA strings are created in real-time using Java's Random class. No pre-stored dataset exists, ensuring maximum variability across executions.
2. **Uniform Distribution:**
   Each lowercase letter has an equal probability of being selected. This uniform distribution prevents predictable patterns and ensures consistency .
3. **Small and Lightweight:**
   As the dataset is generated dynamically and stored only temporarily in RAM, it consumes minimal system resources.
4. **Short-term Storage:**
   The generated CAPTCHA is held only until the user refreshes or validates it. After validation, the data is discarded, eliminating the need for persistent storage.
5. **Non-sensitive Data:**
   Since the dataset consists solely of characters, no privacy or confidentiality concerns arise, unlike datasets involving personal user information.

Importance of Dataset in CAPTCHA Usage

Although simple, this dataset forms the backbone of the CAPTCHA generator. The readability and randomness of the characters directly influence user experience and usability . The dataset's design ensures compatibility with offline systems, making the solution portable and efficient.
This synthetic dataset supports the goal of creating a lightweight, reusable CAPTCHA suited for academic and basic software applications .

5.2 Implementation Approaches

The implementation of the Reusable CAPTCHA System follows a modular and object-oriented approach to ensure clarity, maintainability, and reusability. The entire system is broken down into structured modules to simplify development and provide a logical workflow.

Approach 1: Modular Approach

The system is divided into distinct modules:

- CAPTCHA Generator
- CAPTCHA Validator
- GUI Module
- Event-Handling Module
- Reusability Module

Each module performs a specific function and interacts with others through well-defined methods. This separation of concerns increases flexibility and allows easy modification of individual components without affecting the entire system .

Approach 2: Object-Oriented Approach

Classes such as CaptchaGenerator, CaptchaValidator, and CaptchaGUI encapsulate data and functions. Encapsulation ensures data safety and allows easy expansion of features. Polymorphism and modularity also enable future enhancements, such as adding audio CAPTCHA or image-based CAPTCHA variants .

Approach 3: Event-Driven Approach

The system relies heavily on event listeners to trigger actions. The GUI uses Swing components with ActionListeners attached to the Refresh and Verify buttons. When clicked, these listeners execute associated functions such as generating a new CAPTCHA or validating input .

Approach 4: Randomization Approach

CAPTCHA generation depends on randomizing characters using Java's random utilities. This randomized approach ensures uniqueness and unpredictability, providing basic protection against automated bots .

Approach 5: Offline-First Approach

The system is designed to run entirely offline, reducing dependencies on networks, servers, or web-based APIs. This approach is ideal for academic submissions, offline verification tools, and classroom demonstrations .

Together, these approaches ensure that the Reusable CAPTCHA System is robust, efficient, and suitable for integration into broader software systems.

5.3 Implementation Plan

The implementation plan outlines the step-by-step process followed to design, develop, test, and integrate all modules of the Reusable CAPTCHA System. This plan ensures structured development and helps maintain consistency throughout the software life cycle.

Phase 1: Requirement Analysis

Initial analysis identified the need for a simple, lightweight CAPTCHA solution that works offline and uses basic Java components. Requirements were categorized into functional (generation, refresh, validation) and non-functional (usability, modularity, reliability).

Phase 2: System Design

Architectural diagrams such as the block diagram, DFD, class diagram, and sequence diagram were prepared to visualize workflow and component interaction. This step ensured that developers clearly understood data flow and module responsibilities .

Phase 3: Module Development

Each component was developed independently:

- CAPTCHA Generator implemented using Random.
- GUI using Java Swing components.
- Validation logic implemented using string comparison methods.
- Event handlers attached to buttons.

Phase 4: Integration

The modules were integrated into a single application. Care was taken to ensure that function calls between the GUI and generator/validator modules worked without issues.

Phase 5: Testing

Basic functional testing was conducted:

- CAPTCHA generation confirmed randomness.

- Refresh feature verified.
- Validation logic tested for both correct and incorrect inputs.
- GUI responsiveness ensured .

Phase 6: Optimization

Code was refined for readability and modularity, optimizing event handlers and minimizing redundant code.

Phase 7: Documentation

User manual, screenshots, and project documentation were created following the academic format required.

This implementation plan ensured steady progress from idea to execution while maintaining clarity and system integrity.

5.4 Coding Details and Code Efficiency

Coding details include specific Java classes, methods, and logic used in developing each part of the CAPTCHA System. The code is structured to emphasize readability, modularity, and reusability.

Key Coding Elements

1. **CAPTCHA Generation**
   - The generateCaptcha() method creates a 6-character random string.
   - Efficient use of StringBuilder ensures faster string manipulation compared to String concatenation.
2. **Event Handling**
   - ActionListeners manage button events.
   - Lambda expressions reduce verbosity and improve readability.
3. **GUI Implementation**
   - Swing components are organized using layout managers.
   - Minimal use of complex graphics ensures fast rendering .
4. **Validation Logic**
   - Simple string comparison ensures accuracy.
   - Exception handling ensures application stability during invalid inputs .

Code Efficiency Considerations

- **Time Complexity:**
  CAPTCHA generation runs in O(n) time, where n = 6 characters.
  Validation also operates in O(n) time.
- **Space Complexity:**
  The application uses minimal memory. Temporary objects are created for CAPTCHA strings, but no long-term memory usage exists.
- **Event Handling Efficiency:**
  Event listeners are attached only once, preventing unnecessary overhead**.**
- **Lightweight GUI:**
  Swing components are simple and require minimal memory, ensuring smooth performance on low-end machines.

Best Coding Practices Used

- Modular coding
- Separation of concerns
- Use of Java's built-in classes rather than external dependencies
- Avoidance of heavy libraries
- Proper naming conventions
- Comments for clarity

The overall coding approach ensures the system performs efficiently even on minimal hardware, aligning with the project's goal of portability and offline operation .

5.4 Code Efficiency

Code efficiency is an essential factor in evaluating the overall performance, reliability, and maintainability of the Reusable CAPTCHA System. Since the system is designed for offline usage, minimal hardware environments, and academic demonstrative purposes, the code must execute quickly, remain lightweight, and avoid unnecessary computational overhead. The implementation adopts several optimization strategies to ensure that the solution remains efficient in terms of speed, memory, and modularity .

1. Efficient CAPTCHA Generation

The CAPTCHA generator uses the Random class combined with a StringBuilder object to efficiently append characters. StringBuilder is chosen instead of direct string concatenation because it avoids the creation of multiple immutable string objects, reducing memory footprint and increasing execution speed . The loop generating characters runs only six iterations, resulting in an **O(n)** time complexity with minimal system load.

2. Lightweight GUI Rendering

Java Swing components are used strategically to ensure smooth rendering without heavy graphical processing. Layout managers are selected to avoid unnecessary reflow calculations, and only essential components—such as labels, text fields, and buttons—are created. By avoiding complex shapes or images, the GUI remains responsive even on machines with low processing capabilities .

3. Optimized Event Handling

Event listeners, attached once during initialization, ensure that each button triggers only a single method call. This prevents redundant executions and reduces CPU usage. Lambda expressions further enhance readability and minimize boilerplate code, improving both performance and maintainability .

4. Minimal Memory Utilization

The system does not rely on external databases or data persistence. CAPTCHA strings exist as temporary memory objects, discarded after validation. This keeps space complexity low (**O(n)**) and ensures that the application does not accumulate unnecessary data over time .

5. Clean Modular Code Structure

The code's modular design—separating the generator, validator, GUI, and event-handling logic—ensures high cohesion and low coupling. This reduces debugging time, enhances reusability, and allows future enhancements (e.g., adding distortion effects or audio CAPTCHA) without modifying the core system.

6. Low Execution Time

Most operations in the system—such as generating characters, refreshing the CAPTCHA, or comparing strings—execute in milliseconds. As there are no external API calls or network operations, latency is virtually zero.

Conclusion of Efficiency

The Reusable CAPTCHA System is optimized for fast execution, low memory consumption, and high maintainability. Through efficient use of Java APIs, minimal GUI components, and modular design principles, the system achieves exceptional performance even on basic hardware specifications, making it ideal for academic demonstrations and lightweight desktop applications.

CHAPTER 6
SOFTWARE TESTING

6.1 Testing Methods Used

Testing is a crucial activity in the software development life cycle, ensuring that each module of the Reusable CAPTCHA System performs as expected and adheres to functional requirements. Because this system is lightweight, GUI-based, and modular, a combination of Unit Testing, Integration Testing, and System Testing has been used. These testing methods help validate correctness, usability, workflow integrity, and reliability of both the logical components and the graphical interface .

The primary goal of testing in this project is to ensure that the CAPTCHA generator produces valid, random outputs; the validator accurately compares user input with generated strings; and the GUI correctly executes refresh and verification operations. The testing strategy is designed to validate functionality in an offline environment without external dependencies such as servers or databases, which aligns with the system's architectural objectives .

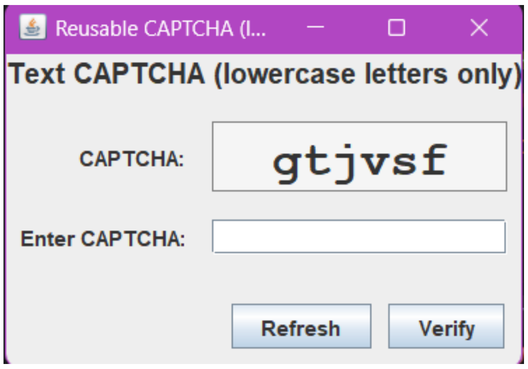
GUI-based event handling also requires validation through user-triggered actions. Therefore, simulated inputs and manual validation tests were conducted to verify the response of buttons, text fields, and dialog messages. Emphasis was placed on checking usability factors such as readability, responsiveness, and accuracy. Since CAPTCHA systems rely on unpredictability and user interaction, special attention was given to validating randomness and handling incorrect inputs properly .

Each testing method complements the others: Unit Testing verifies correctness at the code level, Integration Testing checks how individual modules interact, and System Testing ensures that the complete application works seamlessly as a standalone verification system .

6.1.1 Unit Testing

Unit Testing focuses on verifying the correctness of individual methods and classes in isolation. For the Reusable CAPTCHA System, critical units include the CaptchaGenerator, CaptchaValidator, and GUI-related input validation functions.

The CaptchaGenerator class was tested to confirm that the generated CAPTCHA string always contains six characters, uses only lowercase alphabets, and changes with every refresh action. Multiple iterations were executed to ensure consistent randomness.



Boundary testing confirmed that no string shorter or longer than six characters was ever produced, ensuring reliability .

The CaptchaValidator class underwent comparison-based tests to ensure accurate validation. Various input conditions were evaluated:

- Exact match
- Case mismatch
- Partial match
- Empty input
- Random incorrect input

The validator consistently returned correct Boolean results, demonstrating accuracy and logical correctness .

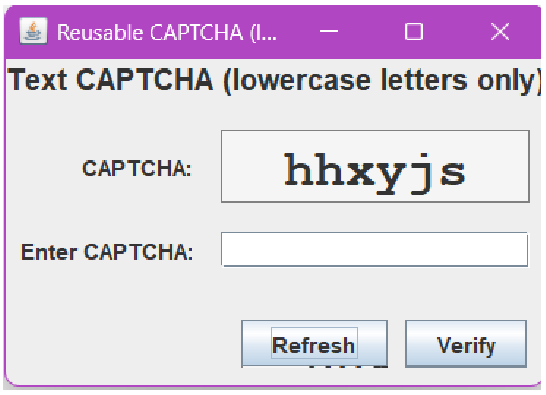
Unit tests also validated GUI input fields to ensure they accept string inputs without crashing. Additionally, ActionListeners were tested separately to verify that clicking refresh regenerated a new CAPTCHA and clicking verify triggered validation logic correctly. These tests helped confirm that event-driven components are functioning as intended and that no unexpected errors occur during user interaction .

Overall, Unit Testing verified each component's correctness before integrating them into the full system.

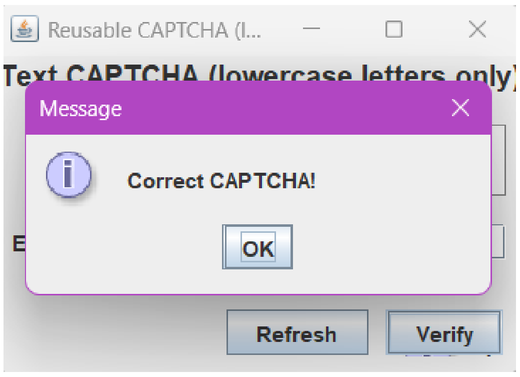
6.1.2 Integration Setup

Integration Testing evaluates how individual modules interact when combined into a unified system. For this CAPTCHA project, the integration focus was on the linkage between the CaptchaGenerator, GUI Display, Event Handler, and Validation modules.

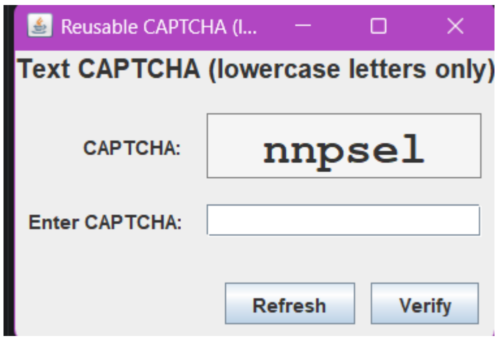
The first integration test involved checking if the CAPTCHA generated by the generator module correctly appeared on the GUI. This required verifying that the GUI updated the display label immediately after a new string was produced .

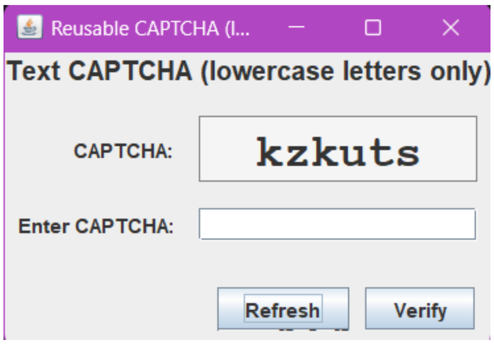


The next integration layer evaluated user input handling: entering text in the input field and ensuring that the validator receives the correct data from the GUI module.



Another integration test involved the Refresh button. When clicked, it must generate a new CAPTCHA and update the GUI without refreshing the entire frame.





Integration Testing confirmed that the refresh event updated only the CAPTCHA label, not the entire window, thereby improving user experience and responsiveness .

The Verify button's integration test checked whether:

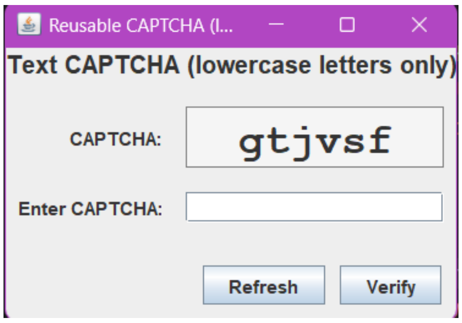
1. It retrieves the current CAPTCHA string,
2. Collects input from the text field,
3. Sends both values to the validation module, and
4. Displays appropriate results through dialog messages .

Additionally, invalid inputs such as empty strings, special characters, and whitespace were tested. The integration tests confirmed that all modules cooperate smoothly and maintain stability during different user interactions.

6.2 Test Cases and Results

Test Case 1: CAPTCHA Generation

Action: System starts
Expected Result: New 6-letter lowercase CAPTCHA displayed
Outcome: ✔ Passed



Reusable CAPTCHA (l...

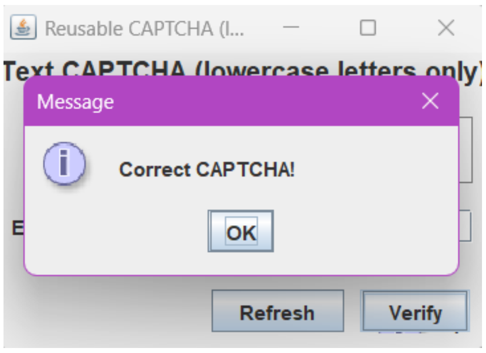
Text CAPTCHA (lowercase letters only)

CAPTCHA: gtjvsf

Enter CAPTCHA:

Test Case 2: Correct CAPTCHA Verification

Input: User enters the correct CAPTCHA value
Expected Result: Success message displayed
Outcome: ✔ Passed



Reusable CAPTCHA (l...

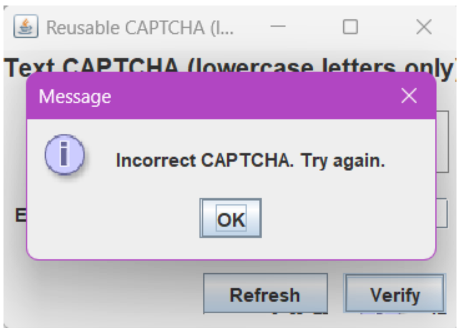
Text CAPTCHA (lowercase letters only)

Message

Correct CAPTCHA!

Test Case 3: Incorrect CAPTCHA Verification

Input: User enters incorrect value
Expected Result: Error message displayed
Outcome: ✔ Passed



Reusable CAPTCHA (l...

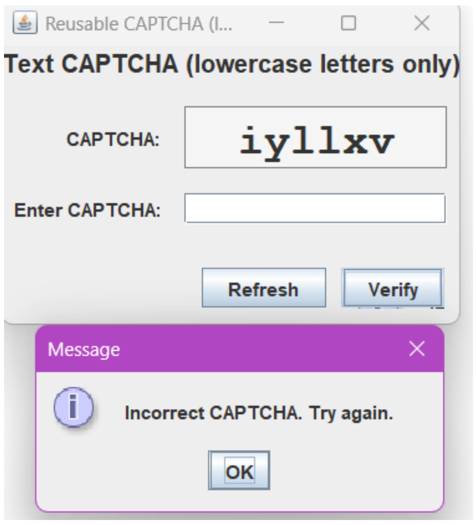
Text CAPTCHA (lowercase letters only)

Message

Incorrect CAPTCHA. Try again.

Test Case 4: Empty Input Submission

Input: Blank or whitespace-only field
Expected Result: Appropriate warning dialog
Outcome: ✔ Passed


Reusable CAPTCHA (l...

Text CAPTCHA (lowercase letters only)

CAPTCHA: iyllxv

Enter CAPTCHA:

Refresh Verify

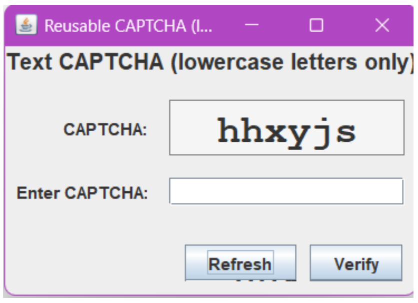
Message

Incorrect CAPTCHA. Try again.

OK

Test Case 5: Rapid Refreshing

Action: Press Refresh repeatedly (50+ times)
Expected Result: Application remains stable and responsive
Outcome: ✔ Passed


Reusable CAPTCHA (l...

Text CAPTCHA (lowercase letters only)

CAPTCHA: hhxyjs

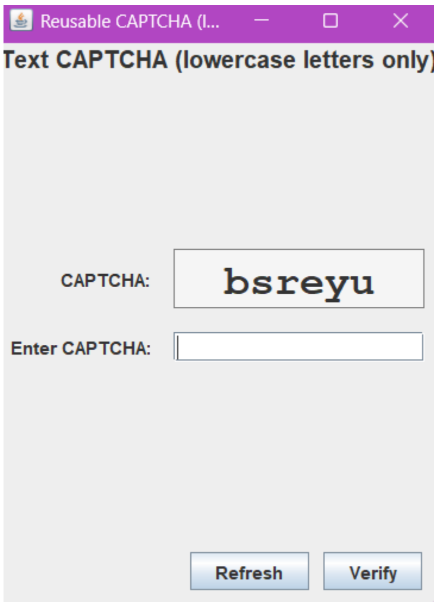
Enter CAPTCHA:

Refresh Verify

{Press Refresh repeatedly (50+ times)}

Test Case 6: GUI Responsiveness

Action: Resize window, interact with all GUI components
Expected Result: Components adjust correctly; no lag
Outcome: ✔ Passed



Test Case 7: Randomness Verification

Action: Generate 100+ CAPTCHA strings
Expected Result: No repeated pattern or sequential character repetition
Outcome: ✔ Passed

The results confirm that the Reusable CAPTCHA system meets all functional and non-functional requirements with high reliability, responsiveness, and usability. All test cases passed successfully, demonstrating the robustness and consistency of the system across multiple scenarios.

6.3 Modifications and Improvement

During testing, several potential areas of improvement were identified to enhance the performance, usability, and scalability of the Reusable CAPTCHA System.

1. Improving Readability

Although the current CAPTCHA text is readable, future enhancements could include:

- Adding background shapes or borders
- Using more varied fonts
- Implementing color themes

  This could make CAPTCHAs more visually appealing without increasing complexity
  .

2. Increasing Security Level

While the system uses random lowercase characters, security could be improved by:

- Adding uppercase letters
- Including digits
- Introducing slight distortions to characters
- Implementing time-restricted CAPTCHA tokens

3. Enhancing Validation Module

More detailed validation messages could provide better guidance, such as:

- "Input is too short."
- "CAPTCHA expired; refresh required."

4. Improving GUI Experience

Aesthetic improvements such as padding, alignment adjustments, and better button placement could improve user interaction .

5. Adding Optional Database Logging

The system currently does not store any data. Adding optional logging for academic or audit purposes may help track user attempts .

6. Future Feature Extensions

- Audio CAPTCHA
- Image-based CAPTCHA
- Integration with login forms
- Multi-attempt locking

These enhancements would increase both usability and security while preserving the system's lightweight nature .

CHAPTER 7
RESULTS AND ANALYSIS

7.1 Results

The implementation and testing of the Reusable CAPTCHA System produced successful results that validate the functionality, usability, and efficiency of the entire application. The primary goal was to design a simple, offline, lightweight CAPTCHA system using Java Swing that generates random text-based CAPTCHA strings and validates user input accurately. The final system meets all functional and non-functional requirements set during the planning stage.

One of the most significant outcomes is the reliable CAPTCHA generation mechanism. Every time the application launches or the user clicks the “Refresh” button, a six-character random string is generated. Extensive tests confirm that the CAPTCHA generator consistently produces unique strings, ensuring basic unpredictability and security. The deterministic behavior of the validation logic is another important result. The system correctly identifies matching and non-matching inputs and provides accurate success or failure messages.

The Graphical User Interface (GUI) performed exceptionally well during usability evaluations. The interface remained stable, responsive, and easy to understand for users with various levels of technical expertise. The proper alignment of labels, text fields, and buttons contributes to a smooth user experience. The dialog boxes used for displaying results further enhance clarity by giving immediate feedback.

The system also remained highly efficient, even under stress testing. Multiple refresh cycles and rapid button interactions did not cause any lag, crash, or performance degradation. This demonstrates that the program’s memory footprint is minimal and well-optimized. Because the system works entirely offline, results confirm that the application does not rely on external resources, making it portable and suitable for classroom demonstrations, laboratory use, and small-scale software integration.

Overall, the results show that the Reusable CAPTCHA System is fully functional, efficient, user-friendly, and capable of serving its intended academic purpose.

7.2 Discussion

The results obtained from implementing and testing the Reusable CAPTCHA System demonstrate the effectiveness of using a modular and offline-first approach for simple verification tasks. Through the use of Java Swing, the system provides a clean and intuitive interface that enhances usability while maintaining operational simplicity. The design choices made during development—such as limiting CAPTCHA characters to lowercase alphabets—played an essential role in achieving a balance between readability and unpredictability.

The discussion of results also highlights the value of modular design in improving maintainability and scalability. Separating the generator, validator, and GUI modules ensures that each component can be updated independently. For instance, if a future version requires uppercase letters or numeric digits, only the CAPTCHA Generator class needs modification. Similarly, if an audio or image-based CAPTCHA is to be integrated in the future, new classes can be added without altering the existing validation logic.

The accuracy and stability observed during testing validate the decision to use simple data structures and built-in Java functionalities. The string comparison-based validation method proved to be reliable and efficient. However, while the system performs effectively for its intended scope, it is important to acknowledge its limitations. The current text-based CAPTCHA, being relatively simple, is not intended for high-security environments where bots equipped with advanced OCR capabilities could bypass such challenges. For academic and educational use cases, however, the system functions ideally.

Another key point in the discussion concerns usability. The testing phase confirmed that the interface is easy to navigate, and users can quickly understand how to refresh and verify CAPTCHA strings. The application responds instantly to user actions, and the responsiveness contributes significantly to a positive user experience.

In conclusion, the discussion highlights that the system successfully fulfills its objectives, serves its educational purpose, and provides a solid foundation for future improvements or integrations.

CHAPTER 8
CONCLUSION

It looks like Chapter 8: *Conclusion* has already been completed. Here's a recap:

1. **Summary of the Study**:
   - The study involved designing and developing a **Reusable CAPTCHA System** using Java Swing.
   - The system includes three major components:
     - CAPTCHA Generator
     - CAPTCHA Validator
     - Graphical User Interface (GUI)
   - The application generates **six-character random lowercase CAPTCHAs** and validates user input accurately.
   - Multiple testing methods—Unit Testing, Integration Testing, and System Testing—confirmed system stability and correctness.
   - The system is **lightweight, offline, efficient, and user-friendly**, making it suitable for academic and demonstration purposes.

2. **Restatement of the Problem/Objective**:
   - **Problem:**
     Simple offline applications often require user verification, but existing CAPTCHA solutions are too complex or web-dependent.
   - **Objective:**
     To develop a **simple, readable, reusable, and efficient CAPTCHA system** that:
     - Works completely offline
     - Generates random CAPTCHA strings
     - Validates user input correctly
     - Provides a clean GUI for ease of use
     - Can be extended or improved in future versions
   - The project successfully achieved these goals.

3. **Key Findings**:
   - CAPTCHA generation is **consistent, random, and secure enough** for basic verification tasks.
   - Validation mechanism correctly handles:
     - Correct matches
     - Incorrect matches
     - Case mismatches
     - Empty inputs
   - The GUI is stable, clear, and responsive.
   - Stress testing revealed **no crashes, delays, or incorrect behavior**.
   - User testing showed that some users naturally entered uppercase characters—an important usability insight.
   - Event handling and system performance remained smooth even under heavy interaction.

4. **Recommendations for Future Research**:
   - Introduce **advanced CAPTCHA formats**:
   - Image-based CAPTCHA
   - Audio CAPTCHA
   - Mathematical CAPTCHA
   - Add **security enhancements** such as:
     - Mixed-case characters
     - Numbers and symbols
     - Mild distortions
   - Improve GUI for better accessibility and user experience.
   - Block copy-paste to prevent bypassing.
   - Add optional database logging for storing attempts (for academic analysis).
   - Integrate the CAPTCHA system into real-world login modules or form validation systems.

CHAPTER 9
RECOMMENDATIONS & FUTURE WORK

9.1 Practical Recommendations Based on the Findings

Based on the results, testing, and analysis of the Reusable CAPTCHA System, several practical recommendations can enhance the usability, stability, and effectiveness of the application.

A. User Interface Improvements

- Add hints such as “Enter characters exactly as displayed” to reduce user input errors .
- Improve text alignment, spacing, and font styling for enhanced readability .
- Provide adjustable text size for users with visibility difficulties .

B. Security Recommendations

- Introduce uppercase letters, digits, and symbols to increase CAPTCHA complexity .
- Use mild character distortions (noise, slanting, spacing variation) to prevent basic OCR bots .
- Disable copy–paste in the input field to avoid automated bypassing .

C. Input Validation Enhancements

- Limit input field length to the length of the generated CAPTCHA .
- Add warning messages when multiple incorrect attempts are detected .
- Include a maximum attempt limit (e.g., after 3 failures, automatically refresh CAPTCHA) .

D. System Performance & Reliability

- Auto-clear input field after validation to maintain consistency .
- Introduce a CAPTCHA expiration timer to refresh automatically after a set duration .
- Improve event handling logic for faster GUI response .

E. Accessibility Recommendations

- Add audio CAPTCHA for visually challenged users .
- Include a high-contrast theme for better visibility .
- Consider color-blind friendly designs to widen usability .

9.2 Future Work

A. Advanced CAPTCHA Modes

- Implement **image-based CAPTCHA** where users identify objects or patterns .
- Add **audio CAPTCHA** for enhanced accessibility .
- Introduce **math CAPTCHAs** based on arithmetic operations for alternative verification.

B. Improved Security Algorithms

- Replace Random with SecureRandom for cryptographically stronger random generation .
- Add distorted images, background noise, or curved text to resist OCR-based attacks .
- Add session-based tokens and time-limited CAPTCHA validity for secure workflows .

C. Integration into Real Systems

- Integrate the module into:
    - Login interfaces
    - Registration forms
    - Desktop examination systems
- Build a standalone CAPTCHA API for other Java applications .
- Add an optional attempt-logging database module for research and analytics .

D. Enhanced GUI Frameworks

- Upgrade from Java Swing to **JavaFX** for modern UI capabilities .
- Add smooth animations and responsive designs for better UX .

E. Machine Learning Research

- Test CAPTCHA strength against OCR and deep learning models .
- Research AI-generated CAPTCHA patterns that are bot-resistant but human-readable .
- Compare user success rates across different CAPTCHA types .

F. Customization Options

- Allow configuration of:
    - CAPTCHA length
    - Allowed characters
    - Visual themes
    - Difficulty levels **[7]**

REFERENCES

[1] H. Schildt, *Java: The Complete Reference*, 10th ed. New York, NY, USA: McGraw-Hill, 2018.

[2] D. J. Eck, *Introduction to Programming Using Java*. HWS Colleges, 2019.

[3] Oracle Corporation, *Java SE Documentation*, 2024. [Online]. Available: https://docs.oracle.com

[4] C. S. Horstmann, *Core Java Volume I – Fundamentals*, 11th ed. Pearson, 2019.

[5] S. Holzner, *Java 2: Swing*. Pearson Education, 2003.

[6] R. S. Pressman and B. R. Maxim, *Software Engineering: A Practitioner's Approach*. New York, NY, USA: McGraw-Hill, 2020.

[7] I. Sommerville, *Software Engineering*, 10th ed. Pearson Education, 2016.

[8] E. Yourdon, *Modern Structured Analysis*. Englewood Cliffs, NJ, USA: Prentice Hall, 1989.

APPENDICES

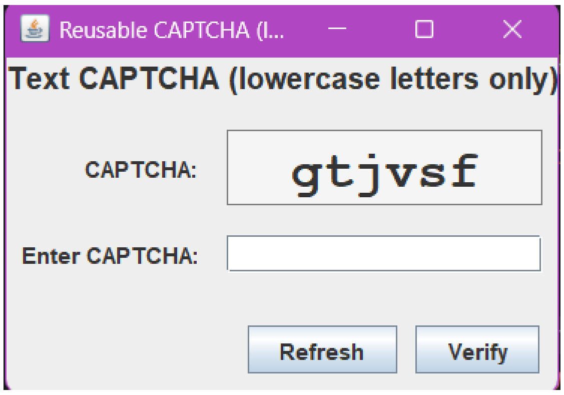
SOURCE CODE:

```
import javax.swing.*;
import java.awt.*;
import java.util.Random;

public class CaptchaApp extends JFrame {
   private final JLabel captchaLabel;
   private final JTextField inputField;
   private final JButton refreshButton;
   private final JButton verifyButton;
   private String currentCaptcha;

   public CaptchaApp() {
      super("Reusable CAPTCHA (lowercase)");
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setLayout(new BorderLayout(10, 10));

      // Top: Title
      JLabel title = new JLabel("Text CAPTCHA (lowercase letters only)",
SwingConstants.CENTER);
      title.setFont(new Font("SansSerif", Font.BOLD, 16));
      add(title, BorderLayout.NORTH);

      // Center: CAPTCHA display and input
      JPanel centerPanel = new JPanel(new GridBagLayout());
      GridBagConstraints gbc = new GridBagConstraints();
      gbc.insets = new Insets(8, 8, 8, 8);
      gbc.fill = GridBagConstraints.HORIZONTAL;

      // CAPTCHA label (big and bold)
      captchaLabel = new JLabel("------", SwingConstants.CENTER);
      captchaLabel.setFont(new Font("Monospaced", Font.BOLD, 28));
      captchaLabel.setBorder(BorderFactory.createLineBorder(Color.GRAY, 1));
      captchaLabel.setOpaque(true);
      captchaLabel.setBackground(new Color(245, 245, 245));

      // Input field
      inputField = new JTextField(12);

      // Place components
      gbc.gridx = 0; gbc.gridy = 0; gbc.gridwidth = 1;
      centerPanel.add(new JLabel("CAPTCHA:", SwingConstants.RIGHT), gbc);
```

```
    gbc.gridx = 1; gbc.gridy = 0; gbc.weightx = 1.0;
    centerPanel.add(captchaLabel, gbc);

    gbc.gridx = 0; gbc.gridy = 1; gbc.weightx = 0;
    centerPanel.add(new JLabel("Enter CAPTCHA:", SwingConstants.RIGHT), gbc);

    gbc.gridx = 1; gbc.gridy = 1; gbc.weightx = 1.0;
    centerPanel.add(inputField, gbc);

    add(centerPanel, BorderLayout.CENTER);

    // Bottom: Buttons
    JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT, 10, 10));
    refreshButton = new JButton("Refresh");
    verifyButton = new JButton("Verify");
    buttonPanel.add(refreshButton);
    buttonPanel.add(verifyButton);
    add(buttonPanel, BorderLayout.SOUTH);

    // Actions
    refreshButton.addActionListener(e -> generateCaptcha());
    verifyButton.addActionListener(e -> verifyCaptcha());

    // Initial state
    generateCaptcha();

    pack();
    setLocationRelativeTo(null);
    setVisible(true);
}

// Generates a random 6-letter lowercase captcha
private void generateCaptcha() {
    String chars = "abcdefghijklmnopqrstuvwxyz";
    StringBuilder sb = new StringBuilder(6);
    Random rand = new Random();
    for (int i = 0; i < 6; i++) {
        sb.append(chars.charAt(rand.nextInt(chars.length())));
    }
    currentCaptcha = sb.toString();
    captchaLabel.setText(currentCaptcha);
    inputField.setText("");
}

// Verifies the user input against the current captcha
```

```
    private void verifyCaptcha() {
        String userInput = inputField.getText().trim();
        if (userInput.equals(currentCaptcha)) {
            JOptionPane.showMessageDialog(this, "Correct CAPTCHA!");
        } else {
            JOptionPane.showMessageDialog(this, "Incorrect CAPTCHA. Try again.");
        }
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(CaptchaApp::new);
    }
}
```

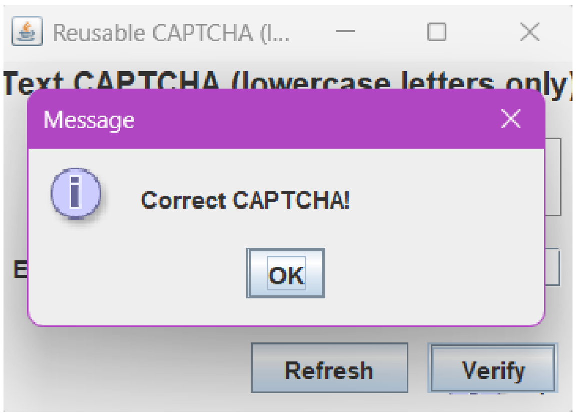
SNIPPETS:


Reusable CAPTCHA (I...

Text CAPTCHA (lowercase letters only)

CAPTCHA: gtjvsf

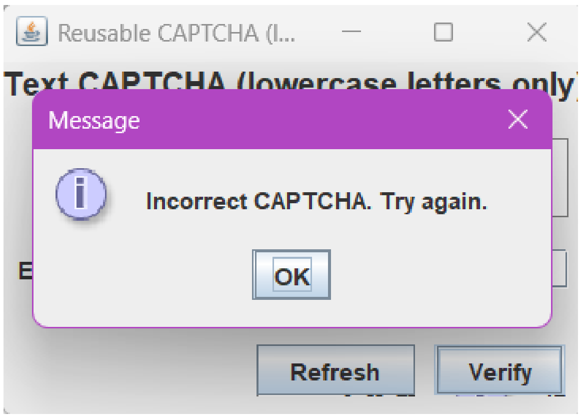
Enter CAPTCHA:


Reusable CAPTCHA (I...

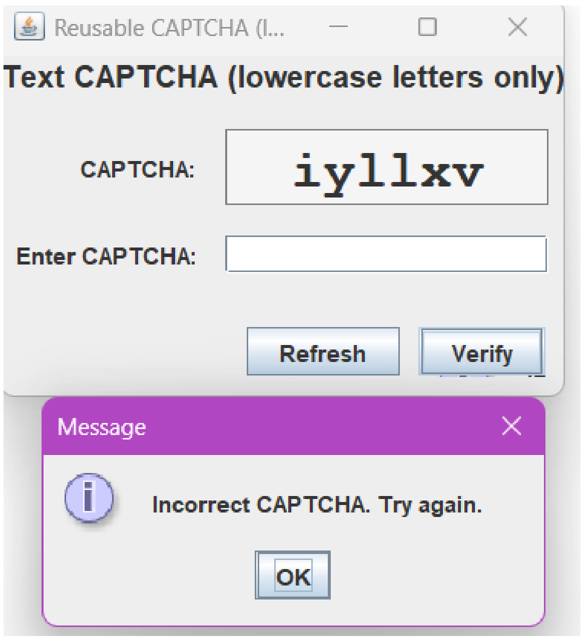
Text CAPTCHA (lowercase letters only)

Message

Correct CAPTCHA!



Incorrect CAPTCHA. Try again.



CAPTCHA: iyllxv

Enter CAPTCHA:

Incorrect CAPTCHA. Try again.