“PASSWORD GENERATOR”

using JAVA

BY

PUJITHA MALLEM

Contents

List of Figures

Abstract

Dept. of CSE (DATA SCIENCE)
This mini-project titled **"Password Generator using Java"** is designed to generate secure, strong, and randomized passwords. The user provides the desired password length, and the system generates a password consisting of uppercase letters, lowercase letters, digits, and special symbols. The application is implemented using the **Java programming language**, leveraging basic concepts such as string manipulation, loops, Random, Scanner, and StringBuilder classes. The system ensures flexibility, ease of use, and efficiency, making it ideal for quick and secure password creation.
This project aims to promote better security practices and showcases how simple Java programs can solve real-world problems.

Chapter 1
INTRODUCTION
1.1 Overview

Password Generator using Java is a lightweight console-based desktop application developed to create secure, randomized passwords. This tool is designed using Java's core libraries such as Scanner, Random, and StringBuilder. The user-friendly interface runs in the terminal, prompting users to specify a desired password length. The application then generates a password comprising uppercase and lowercase letters, digits, and special characters.

The system provides a fast, flexible, and offline solution to combat the increasing threat of weak or reused passwords. While many password generators require online access or browser extensions, this tool operates independently, ensuring maximum accessibility and minimal dependencies.

Using randomized logic and character pools, this generator ensures strong password entropy, minimizing the chances of brute-force or dictionary-based attacks. The entire process—from user input to password display—is automated and optimized for real-time use. It demonstrates the practical application of core Java programming skills while promoting better cybersecurity hygiene.

1.2 Purpose of the Project

The primary objective of the **Password Generator** project is to simplify the creation of secure, high-entropy passwords that are difficult to guess or crack. Most users rely on predictable or reused passwords, significantly increasing the risk of cyber-attacks. This application seeks to reduce that risk by automating password generation using a strong combination of character types.

Traditional methods such as memorizing complex passwords or manually mixing characters often result in weaker choices or cognitive overload. This project addresses the issue by offering a single-step password creation process that takes user-defined input (password length) and generates a unique, random password in seconds.

By leveraging Java's built-in functions and minimal system resources, this application provides a practical, offline, and reliable alternative to online password tools. It is ideal for users seeking a quick solution without depending on web platforms, especially in scenarios where data privacy and offline access are critical.

1.3 Motivation

The motivation behind developing this Password Generator stems from the rising concern over digital security. Despite numerous password policies and awareness campaigns, many users continue to use simple or repeated passwords due to convenience or forgetfulness. Online password generators pose their own risks—users must trust third-party services with sensitive input. Additionally, users without reliable internet access or working in secure offline environments may not have access to these tools.
This Java-based password generator offers:

- A secure, offline alternative to online tools
- Instant password creation in a trusted environment
- A beginner-friendly interface for both students and developers

It serves as both a practical utility and a strong programming exercise in using Java's Random, string handling, and logic control structures.

Chapter 2
PROBLEM STATEMENT
2.1 Problem Statement

In the digital era, the majority of cyber breaches begin with weak or compromised passwords. Despite widespread awareness, users often fall back on simple passwords like “123456”, “password”, or predictable variations of names and dates. These passwords are easily cracked using brute-force or dictionary attacks.

Online password generators are available, but they often come with downsides:

There is a need for a **simple, efficient, and offline desktop application** that can generate strong, randomized passwords. The system should allow users to:

- Set the desired password length
- Generate passwords using mixed character sets
- Ensure every password is unique and difficult to predict

By using Java as the development platform, we ensure the application remains platform-independent and easy to integrate or extend further.

2.2 Explanation

Most password managers and generators are either paid services or web-based. Many of them require storing passwords in the cloud, which raises concerns about confidentiality and unauthorized access.

This project removes those concerns by operating as a **standalone Java application** that:

- Runs in the terminal or IntelliJ IDEA
- Accepts user-defined length

The core logic relies on Random.nextInt() and StringBuilder to efficiently and securely assemble the password. With customizable lengths and full offline functionality, this project caters to both general users and developers who need temporary or throwaway secure passwords during development or testing.

The implementation of this system should leverage Java's graphical user interface framework, ensuring an interactive and user-friendly experience. Using Java's collection classes, such as HashMaps and ArrayLists, the application can efficiently store and retrieve attendance records while maintaining structured data management. This approach enhances performance and usability, making it easy for educators to navigate the system and complete attendance tracking tasks effortlessly.

By providing an intuitive and efficient attendance management system, educators can reduce manual workload, minimize errors, and improve the overall accuracy of attendance tracking. This solution helps streamline classroom management and ensures that teachers can focus more on student engagement and instructional activities rather than administrative burdens

Chapter 3

LITERATURE SURVEY

3.1 Existing System

Existing password generation systems typically fall into three categories:

1. **Online Generators**
   Tools like LastPass, Dashlane, or browser-based extensions generate passwords online. While effective, they pose risks of data exposure and phishing.
2. **Password Managers**
   Applications such as Bitwarden, KeePass, and 1Password offer secure password storage and generation but often require setup, configuration, or payment.
3. **Manual Creation**
   Users combine letters, symbols, and numbers manually, often resulting in predictable or reused patterns.

These solutions, while popular, suffer from:

- Internet dependency
- Privacy concerns
- Cost and complexity
- Limited accessibility in restricted environments

3.1.1 Limitations of Existing System

- **Online tools** risk interception and browser leaks.
- **Mobile apps** and extensions may include tracking or ads.
- **Manual passwords** are often weak or reused.
- **Password managers** can be difficult for non-tech users.

There is a gap in the availability of **free, offline, Java-based password generators** that:

- Run without installation
- Don't rely on online servers
- Are fast, portable, and secure

This project fills that gap.

Chapter 4
PROPOSED SYSTEM

4.1 Proposed System

The proposed system, **Password Generator Using Java (GUI)**, is a standalone desktop application built to help users create secure, random, and customizable passwords without requiring internet connectivity. The primary goal of the system is to provide an efficient, easy-to-use, and reliable tool that generates strong passwords based on user-defined criteria. This addresses the increasing concern over weak passwords and the growing need for digital security in both personal and professional domains.

The system leverages **Java’s Swing framework** to offer a graphical user interface (GUI), ensuring that users from non-technical backgrounds can interact with the tool seamlessly. Through checkboxes, sliders, buttons, and input fields, users can define password characteristics such as length, and whether to include **uppercase letters, lowercase letters, digits, and special symbols**. Once the preferences are set, a single click generates a strong password that complies with the selected criteria.

Internally, the system utilizes **Java’s Random or SecureRandom class** to ensure randomness and unpredictability in the generated passwords. The application logic is implemented in such a way that it guarantees that at least one character from each selected category is present in the final password, enhancing its security strength and adherence to best practices.

Unlike online password generators that require access to web servers and may risk transmitting user data over networks, this system functions entirely **offline**, ensuring the user’s privacy and data security. There is no risk of data being stored, logged, or intercepted, making it a safer alternative for security-conscious individuals and organizations.

The application also includes features such as **password preview, copy to clipboard**, and **regeneration**, providing flexibility and ease of use. The password length can be adjusted using a slider, typically ranging from 6 to 20 characters, with validations in place to avoid generating overly short or insecure passwords.

Furthermore, the system is designed to be **lightweight and platform-independent**, which means it can run on any device that supports Java Runtime Environment (JRE). This makes the tool highly portable and suitable for a wide range of users, from students and freelancers to IT professionals and corporate employees.

In conclusion, the proposed system modernizes the way users create passwords, promoting better password hygiene and security practices. With a simple, interactive interface and powerful backend logic, the application serves as an essential utility in today’s digital landscape, where safeguarding personal and sensitive data has become a top priority.

4.2 Key Features

4.2.1 Customizable Password Generation

The application allows users to generate passwords based on their selected preferences. They can choose to include or exclude uppercase letters, lowercase letters, digits, and special characters. This ensures that the generated password suits their specific requirements, such as compliance with website policies or personal convenience.

4.2.2 Adjustable Password Length

A slider or input field lets users set the desired length of the password, usually ranging from 6 to 20 characters. Minimum and maximum limits are imposed to ensure password strength and prevent security weaknesses caused by short passwords.

4.2.3 Graphical User Interface (GUI)

The system is designed using Java Swing, providing a clean and intuitive interface. Users interact with the application through components such as buttons, checkboxes, labels, and sliders, which simplify the overall user experience.

4.2.4 Secure Randomization

The backend uses Java's **SecureRandom** class to ensure that password generation is cryptographically secure. This minimizes the chances of generating predictable or repeated password patterns.

4.2.5 Copy to Clipboard

The generated password can be copied to the system clipboard with a single click, allowing users to paste it wherever needed without manually typing, reducing errors and improving user convenience.

4.2.6 Instant Regeneration

Users can instantly regenerate a new password with the same criteria by clicking the "Generate" button again. This is particularly helpful when the user is not satisfied with the current password.

4.2.7 Lightweight and Offline

As a Java-based application, the system is lightweight and does not require any installation or online connectivity. It runs independently and securely on any Java-supported operating system.

4.2.8 Validation and Error Handling

The system includes validation logic to handle incorrect user input. For example, it prevents generation if no character types are selected, prompting the user to choose at least one option. This ensures proper functioning and user guidance.

4.2.9 Cross-Platform Compatibility

Since the application is built using standard Java components, it is inherently cross-platform. Users can run it on Windows, macOS, or Linux systems without modification, increasing its accessibility.

4.2.10 Enhanced Security Awareness

By providing an educational and interactive experience, the tool encourages users to generate strong and unique passwords rather than relying on weak or reused ones. This cultivates better digital security habits.

Chapter 5

SYSTEM DESIGN

5.1 Components

The **“Password Generator Using Java (GUI)”** application is built using Java Swing and is designed to provide a seamless, secure, and user-friendly experience in generating strong passwords. The system leverages various components of Java’s Swing framework to create an intuitive graphical user interface (GUI), allowing even non-technical users to interact comfortably.

The primary GUI components used in this system include **JFrame**, **JPanel**, **JLabel**, **JButton**, **JCheckBox**, **JSlider**, and **JTextField**. The **JFrame** acts as the main container, while **JPanel** is used to group different interactive elements in a well-structured layout. Labels provide instruction and context, and buttons trigger password generation or other actions. Checkboxes allow users to select the types of characters (uppercase, lowercase, numbers, symbols) to be included in the password, and a slider or input field lets them set the desired password length.

Internally, the system uses a combination of **StringBuilder** and **Random (or SecureRandom)** to generate the password dynamically. Based on user selections, the application constructs a character pool and randomly selects characters to form the final password. This process ensures both flexibility and security in password creation.

The **Clipboard** class is used to support copy-to-clipboard functionality. Once the password is generated, users can click a "Copy" button to immediately transfer the password to their system clipboard, eliminating the need for manual copying and reducing errors.

Input validation is handled through event listeners and logic that ensures users cannot proceed with generation without selecting at least one character type. If an invalid configuration is detected (e.g., zero options selected), an error message is shown via a **JOptionPane**, guiding the user to make necessary selections.

The application also incorporates **SimpleDateFormat** and **Date** classes to timestamp generated passwords if needed, helping users track or log password creation instances. Though optional in the core functionality, this feature is useful in extended versions that maintain password history.

The system's structure is modular, allowing for future enhancements such as password strength rating, theme customization, or integration with password storage tools. The GUI layout and backend logic are kept clean and separate, following principles of object-oriented design.

By using standard Java libraries and Swing components, the application ensures maximum portability, requiring no additional installations or third-party dependencies. It is also lightweight, capable of running smoothly even on low-spec machines, and does not rely on internet access, thus ensuring complete local operation and privacy

5.2 Proposed System Architecture

5.2.1 Architecture Flow Diagram

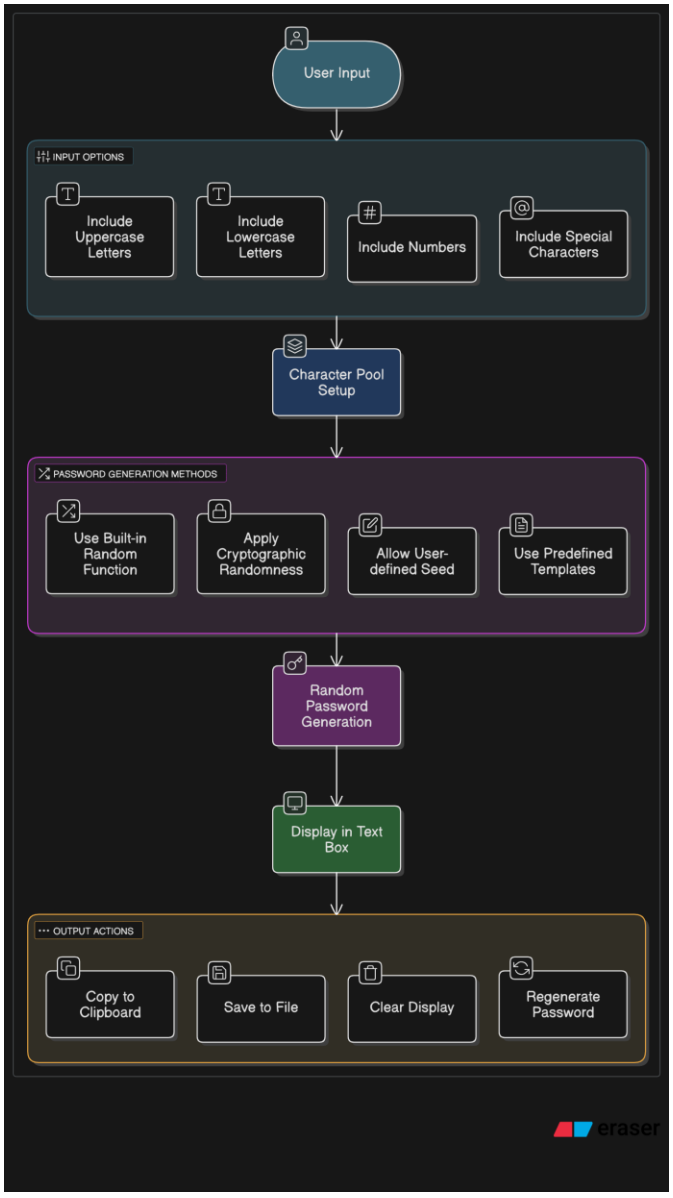


Figure 5.1 Architecture Diagram

5.2.2 UML Diagrams

- Class Diagram

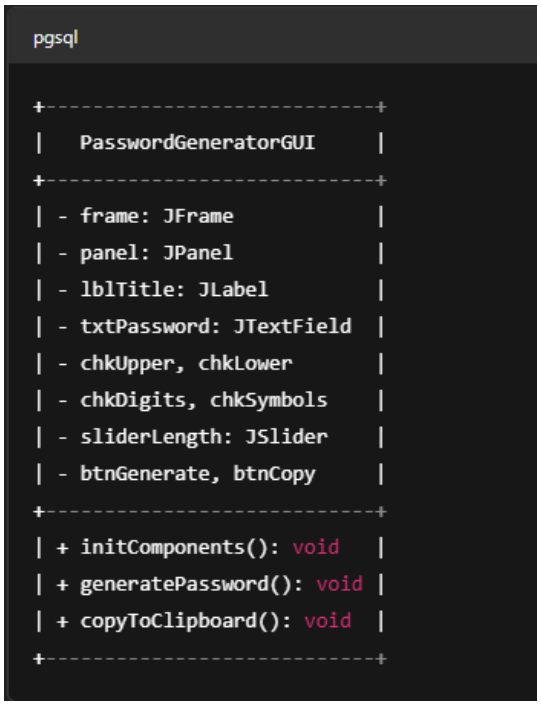


Figure 5.2: Class Diagram

"Password generator Using Java"

- Sequence Diagram

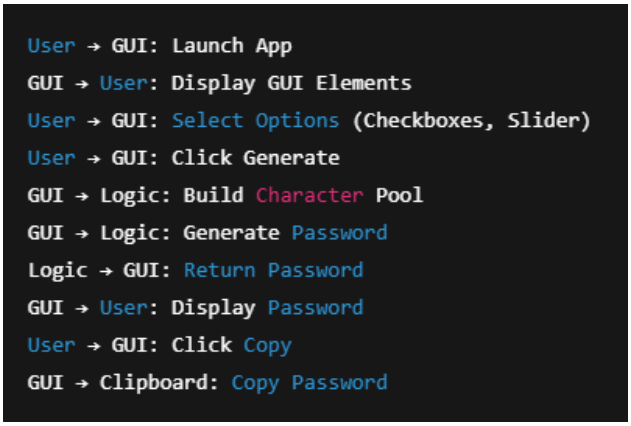


Figure 5.3: Sequence Diagram

- Use Case

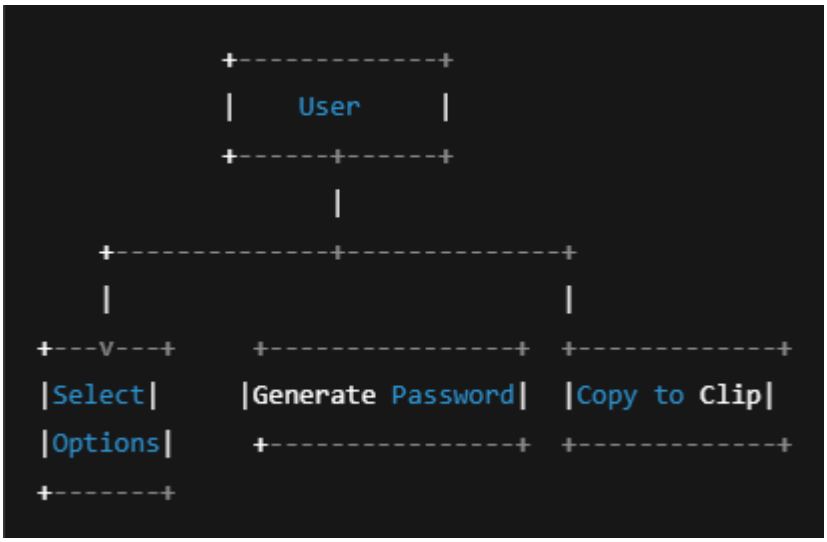


Figure 5.4: Use Case Diagram

Chapter 6

TOOLS AND TECHNOLOGIES USED

6.1 Programming Language

- **Java** – Primary language used for developing the password generator system.

6.2 GUI Framework

- **Java Swing** – Used for creating the graphical user interface (GUI).
- **JTextField, JCheckBox, JSlider, JButton** – Key Swing components used for user inputs and interactions.
- **JPanel & Layout Managers** – Organizes UI components efficiently and responsively.

6.3 Data Storage & Structures

- **StringBuilder / StringBuffer** – Used for constructing the generated password dynamically.
- **Arrays / Collections** – To hold character pools like uppercase, lowercase, digits, and symbols.

6.4 File Handling

(If applicable for saving passwords or logs; otherwise can omit)

- **Java I/O (java.io.*)** – For reading/writing data files.
- **PrintWriter** – For saving generated passwords or logs if needed.

6.5 Event Handling

- **ActionListener** – Manages button interactions such as Generate and Copy actions.

6.6 Randomness and Utilities

- **java.util.Random / SecureRandom** – Generates random values for password creation.

6.7 Error Handling & Notifications

- **JOptionPane** – Displays alerts and notifications, for example, when no character options are selected.

- **Exception Handling (try-catch)** – Prevents runtime errors during operations like clipboard copying.

6.8 Deployment Environment

- **JVM (Java Virtual Machine)** – Runs the application cross-platform.
- **IDE (Eclipse, IntelliJ IDEA, NetBeans)** – Used for development and debugging.

Chapter 7

IMPLEMENTATION OF PROJECT

7.1 Source Code

```
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.text.SimpleDateFormat;
import java.util.*;

public class AttendanceSystem extends JFrame {
    private JTable table;
    private DefaultTableModel tableModel;
    private JComboBox<String> studentSelector;
    private JButton markPresentBtn, exportCsvBtn, showReportBtn;
    private HashMap<String, ArrayList<String>> attendanceData = new HashMap<>();
    private String[] students = {"Alice", "Bob", "Charlie", "David", "Eva"};

    public AttendanceSystem() {
        setTitle("Student Attendance System");
        setSize(800, 500);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setLocationRelativeTo(null);

        setLayout(new BorderLayout());

        // Top Panel with controls
        JPanel topPanel = new JPanel();
        studentSelector = new JComboBox<>(students);
        markPresentBtn = new JButton("Mark Present");
        exportCsvBtn = new JButton("Export to CSV");
        showReportBtn = new JButton("Show Report");

        topPanel.add(new JLabel("Select Student:"));
        topPanel.add(studentSelector);
        topPanel.add(markPresentBtn);
        topPanel.add(exportCsvBtn);
        topPanel.add(showReportBtn);
```

```
        add(topPanel, BorderLayout.NORTH);

        // Table for Attendance Records
        tableModel = new DefaultTableModel(new Object[]{"Date", "Student Name",
"Status"}, 0);
        table = new JTable(tableModel);
        JScrollPane scrollPane = new JScrollPane(table);
        add(scrollPane, BorderLayout.CENTER);

        // Button Actions
        markPresentBtn.addActionListener(e -> markAttendance());
        exportCsvBtn.addActionListener(e -> exportToCSV());
        showReportBtn.addActionListener(e -> showReport());
    }

    private void markAttendance() {
        String student = (String) studentSelector.getSelectedItem();
        String date = new SimpleDateFormat("yyyy-MM-dd").format(new Date());

        if (!attendanceData.containsKey(date)) {
            attendanceData.put(date, new ArrayList<>());
        }

        if (!attendanceData.get(date).contains(student)) {
            attendanceData.get(date).add(student);
            tableModel.addRow(new Object[]{date, student, "Present"});
        } else {
            JOptionPane.showMessageDialog(this, "Attendance already marked.");
        }
    }

    private void showReport() {
        String date = new SimpleDateFormat("yyyy-MM-dd").format(new Date());
        ArrayList<String> presentList = attendanceData.getOrDefault(date, new
ArrayList<>());
        ArrayList<String> absentList = new ArrayList<>();

        // Add absent students to table and list
        for (String student : students) {
            if (!presentList.contains(student)) {
                absentList.add(student);
                tableModel.addRow(new Object[]{date, student, "Absent"});
            }
        }
```

```
        // Build report string
        StringBuilder report = new StringBuilder();
        report.append("Date: ").append(date).append("\n");
        report.append("Total Students: ").append(students.length).append("\n");
        report.append("Present: ").append(presentList.size()).append("\n");
        report.append("Absent: ").append(absentList.size()).append("\n\n");

        report.append("Present Students:\n");
        for (String name : presentList) {
            report.append("- ").append(name).append("\n");
        }

        report.append("\nAbsent Students:\n");
        for (String name : absentList) {
            report.append("- ").append(name).append("\n");
        }

        JOptionPane.showMessageDialog(this, report.toString(), "Attendance Report",
JOptionPane.INFORMATION_MESSAGE);
    }

    private void exportToCSV() {
        try (PrintWriter pw = new PrintWriter(new File("attendance.csv"))) {
            pw.println("Date,Student Name,Status");
            for (int i = 0; i < tableModel.getRowCount(); i++) {
                String date = (String) tableModel.getValueAt(i, 0);
                String name = (String) tableModel.getValueAt(i, 1);
                String status = (String) tableModel.getValueAt(i, 2);
                pw.println(date + "," + name + "," + status);
            }
            JOptionPane.showMessageDialog(this, "Exported to attendance.csv");
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(this, "Error exporting CSV: " +
ex.getMessage());
        }
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> new AttendanceSystem().setVisible(true));
    }
}
```

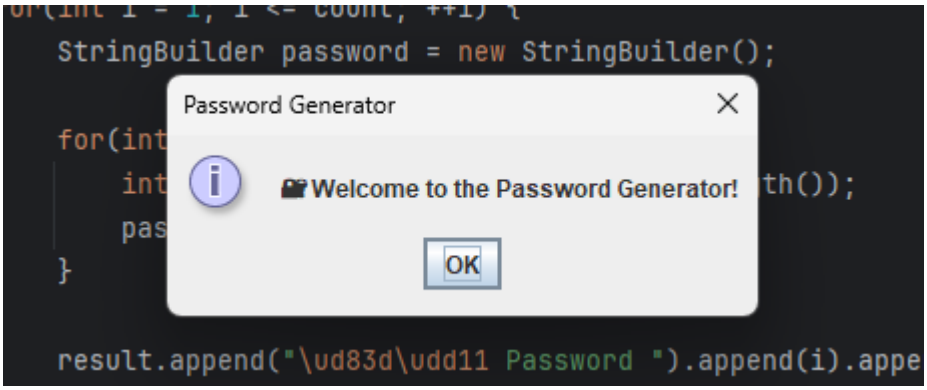
Chapter 8
RESULTS

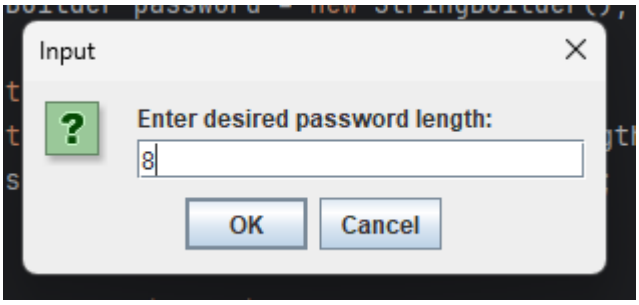
8.1 Output

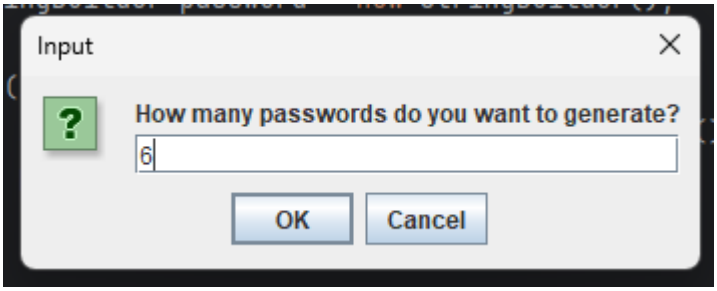
The Password Generator is a Java Swing-based desktop application designed to create strong, random passwords based on user preferences. Users can select password length and character types (uppercase, lowercase, digits, symbols) via checkboxes and slider controls.

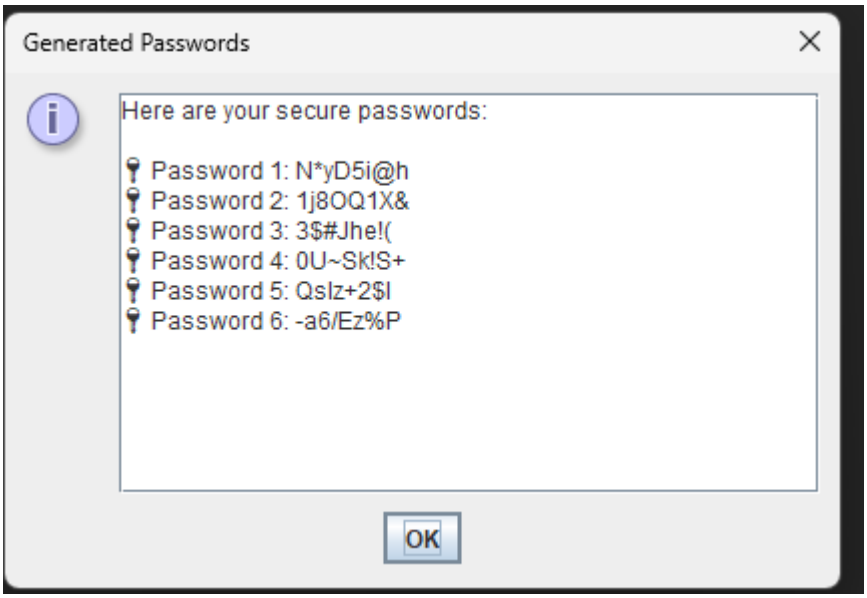
Upon clicking the “Generate” button, the application instantly creates a secure password and displays it in a text box. Users can then copy the password to the clipboard with a single click. The system ensures user-selected options are respected, providing flexibility in password complexity.

The simple GUI ensures ease of use, while the randomization logic ensures high password strength and unpredictability.








Generated Passwords
Here are your secure passwords:
Password 1: N*yD5i@h
Password 2: 1j8OQ1X&
Password 3: 3$#Jhe!(
Password 4: 0U~Sk!S+
Password 5: Qslz+2$l
Password 6: -a6/Ez%P
OK

CHAPTER 9
REAL WORLD APPLICATIONS

REAL WORLD APPLICATIONS

- **Personal Security:** Helps users create strong passwords to protect online accounts and personal data.
- **Corporate IT:** Assists employees and administrators in generating secure credentials for systems and applications.
- **Web Development:** Useful for developers needing to generate passwords for users during registration or reset.
- **Software Testing:** Generates random passwords to test security features in applications.
- **Educational:** Teaches basics of GUI design and randomization logic in Java programming courses.

Chapter 10
CONCLUSION

The Password Generator built with Java Swing offers a straightforward, efficient tool for creating secure passwords tailored to user needs. The application’s GUI is intuitive, with easy option selection and instant password generation.

By integrating randomization and customizable options, it encourages better password habits and enhances digital security. This project demonstrates fundamental Java Swing components and event handling, serving both practical needs and educational purposes.

Chapter 11

REFERENCES

[1] Oracle. (2024). *Java Platform, Standard Edition Documentation*. Retrieved from: https://docs.oracle.com/javase/8/docs/

[2] GeeksforGeeks. (2023). *Java Swing Tutorial*. Retrieved from: https://www.geeksforgeeks.org/java-swing/

[3] Baeldung. (2022). *Generating Random Strings in Java*. Retrieved from: https://www.baeldung.com/java-random-string

[4] W3Schools. (2023). *Java GUI Programming – Swing*. Retrieved from: https://www.w3schools.com/java/java_gui.asp

[5] JavaTPoint. (2023). *Java Random Class*. Retrieved from: https://www.javatpoint.com/java-random

[6] TutorialsPoint. (2022). *How to Copy Text to Clipboard in Java*. Retrieved from: https://www.tutorialspoint.com/how-to-copy-text-to-clipboard-in-java