FOCUS BUDDY

A Project Report Submitted in the partial fulfillment of the requirements of the course titled

"Problem Solving Through Programming (JAVA)"

BACHELOR OF TECHNOLOGY

In

DEPARTMENT OF FRESHMAN ENGINEERING

By

M SRI NITHYA	**2520030447**
G JAHNAVI	**2520030483**
ANANYA ROY	**2520030471**
Y EEKSHITHA	**2520030615**
M MANASA	**2520030500**
B HAASINI	**2520030265**

Under the Esteemed Guidance of

Guide Name **Dr Jayaram Boga**

Assistant Professor

Department of Freshman Engineering

Koneru Lakshmaiah Education Foundation

(Deemed to be University estd. u/s. 3 of the UGC Act, 1956)
Off-Campus: Bachupally-Gandimaisamma Road, Bowrampet, Hyderabad, Telangana - 500 043.
Phone No: 7815926816, www.klh.edu.in

K L (Deemed to be) University

DEPARTMENT OF FRESHMAN ENGINEERING

Declaration

The Project Report entitled **" FOCUS BUDDY"** is a record of Bonafide work of **M SRI NITHYA-2520030447, G JAHNAVI-2520030483, ANANYA ROY-2520030471, Y EEKSITHA-2520030615, M MANASA-2520030500, B HAASINI-2520030265** submitted in partial fulfillment of the requirements of the course titled "Problem Solving Through Programming (JAVA)" under the B.Tech Ist Year Trimester - I program in Department of Freshman Engineering at K L University. The results presented in this report have not been copied from any other department, university, or institute.

M SRI NITHYA – 2520030447

G JAHNAVI – 2520030483

ANANYA ROY – 2520030471

Y EEKSHITHA – 2520030615

M MANASA – 2520030500

B HAASINI – 2520030265

K L (Deemed to be) University

DEPARTMENT OF FRESHMAN ENGINEERING

CERTIFICATE

This is to certify that the java project based report entitled **"FOCUS BUDDY"** is a bonafide work done and submitted by **M SRI NITHYA-2520030447,G JAHNAVI-2520030483, ANANYA ROY-2520030471, Y EEKSITHA-2520030615, M MANASA-2520030500, B HAASINI-2520030265** in partial fulfillment of the requirements of the course titled "Problem Solving Through Programming (JAVA)" under the B.Tech Ist Year Trimester - I program in Department of Freshman Engineering, K L (Deemed to be University), during the academic year **2025-2026.**

Signature of the Guide

Signature of the Course Coordinator **Signature of the HOD**

ACKNOWLEDGEMENT

The success in this project would not have been possible but for the timely help and guidance rendered by many people. Our wish to express my sincere thanks to all those who has assisted us in one way or the other for the completion of my project.

Our greatest appreciation to my Course Coordinator **Dr Y Ashok**, and my guide **Dr Jayaram Boga,** Department of Freshman Engineering which cannot be expressed in words for his/her tremendous support, encouragement and guidance for this project.

We express our gratitude to **Dr. N. Chaitanya Kumar**, Head of the ***Department for Freshman Engineering*** for providing us with adequate facilities, ways and means by which we are able to complete this project-based Lab.

We thank all the members of teaching and non-teaching staff members, and also who have assisted me directly or indirectly for successful completion of this project. Finally, I sincerely thank my parents, friends and classmates for their kind help and cooperation during my work.

M SRI NITHYA – 2520030447
G JAHNAVI – 2520030483
ANANYA ROY – 2520030471
Y EEKSHITHA – 2520030615
M MANASA – 2520030500
B HAASINI – 2520030265

ABSTRACT

Maintaining focus and productivity has become increasingly challenging in today's digital environment due to constant distractions, unstructured study habits, and lack of personal discipline. Focus Buddy is a smart study timer developed in Java that aims to improve concentration by implementing the Pomodoro technique, which breaks study sessions into well-defined intervals followed by short breaks. This structured approach helps users stay consistent, avoid mental fatigue, and develop a sustainable study routine. To encourage motivation alongside productivity, the system displays simple motivational quotes during breaks, helping users stay mentally positive and reducing procrastination. The application includes basic controls such as start, pause, and reset, which allow users to manage their study sessions easily. The interface is straightforward and minimal, making it easy for beginners to use without the need for complex instructions or additional technical knowledge. Research in cognitive science suggests that periodic breaks improve memory retention, focus levels, and long-term performance. By integrating these principles into a basic desktop application, Focus Buddy offers a practical solution for improving academic discipline and reducing stress. Overall, the system promotes healthy study patterns, supports mental well-being, and acts as a helpful companion for students working toward better productivity.

INDEX

CHAPTER -1 INTRODUCTION

1.1 Background of the Project

Time management has become one of the most essential skills in modern academic and professional environments. Students and workers alike are frequently challenged by distractions, overwhelming workloads, and the difficulty of maintaining focus for extended periods. As digital technology rapidly evolves, individuals often find themselves surrounded by numerous electronic distractions, social media notifications, entertainment platforms, and online communication tools that disrupt concentration and productivity. To address these challenges, productivity techniques have been developed to help individuals work efficiently and effectively.

As the technique gained popularity, numerous digital timers and applications emerged to assist users in implementing the Pomodoro approach.

1.2 Problem Statement

Students often struggle with maintaining focus and managing their study time effectively. With the rise of digital distractions and multitasking tendencies, it has become increasingly difficult for learners to stay committed to long periods of studying. Extended sessions without breaks can lead to mental fatigue, procrastination, stress, and reduced learning efficiency.

While various Pomodoro applications already exist, many of them come with limitations:

- They may require internet access or user accounts.
- Some contain advertisements or distracting elements.
- Many do not offer adequate customization of work and break durations.
- Others lack simplicity or present overly complex interfaces that overwhelm users.

- Some mobile applications drain device batteries or have intrusive notifications.

These issues highlight the need for a clean, efficient, and customizable Pomodoro study timer developed specifically with student productivity in mind.

Thus, the problem at hand is the lack of a simple and effective study timer tool that supports focused study sessions, enhances time management skills, and helps users maintain discipline while avoiding distractions.

Purpose of the Project

The primary purpose of this project is to develop a Pomodoro Study Timer application that assists students in managing their study sessions and improving productivity. The tool aims to provide a structured approach to studying by implementing the Pomodoro Technique in a user-friendly and efficient software environment.

More specifically, the project aims to:

1. Implement a reliable countdown timer.

2. Allow users to customize their work and break durations.

3. Provide visual and audio notifications to guide transitions between sessions.

4. Help students build consistent study habits through structured time intervals.

5. Offer features such as session tracking, progress display, or task labeling.

By achieving these goals, the timer serves as a personal productivity assistant that enhances concentration, reduces procrastination, and fosters a healthy study routine.

Related Work and Existing Systems

Applications such as Focus Booster, Forest, Marinara Timer, and Tomighty are widely used Pomodoro tools. However, these systems have certain drawbacks:

- Some require payment for advanced features.
- Many include advertisements or unnecessary animations.
- Others lack customization options.
- Several rely on internet connectivity.
- Many apps are designed for mobile and not for desktop use.

The proposed Pomodoro Study Timer differentiates itself by being:

- Lightweight and offline.
- Simple and distraction-free.
- Fully customizable.
- Ideal for academic use.

Objectives of the Study

General Objective:

To develop an easy-to-use Java application that implements the Pomodoro Technique to help students improve their study habits and productivity.

Specific Objectives

1. To design a user-friendly interface that simplifies time management.

2. To implement an accurate countdown timer.

3. To allow full customization of study and break intervals.

4. To incorporate optional sound and visual notifications.

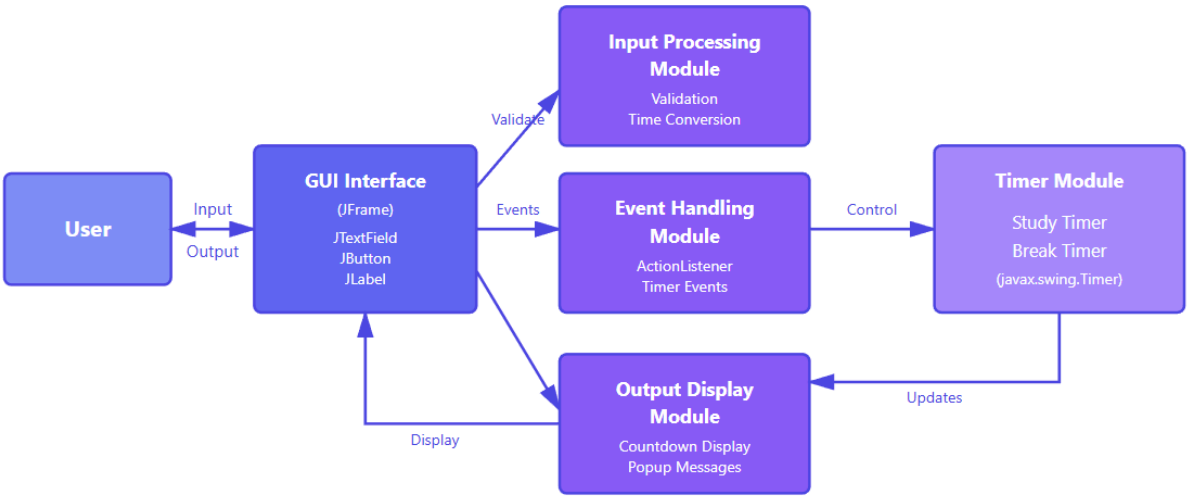
5. To ensure stable performance across various operating systems.

6. To store user preferences for a more personalized experience.

7. To provide basic session tracking to motivate consistent study.

CHAPTER -2 SYSTEM ARCHITECTURE

2.1 High-level architecture diagram

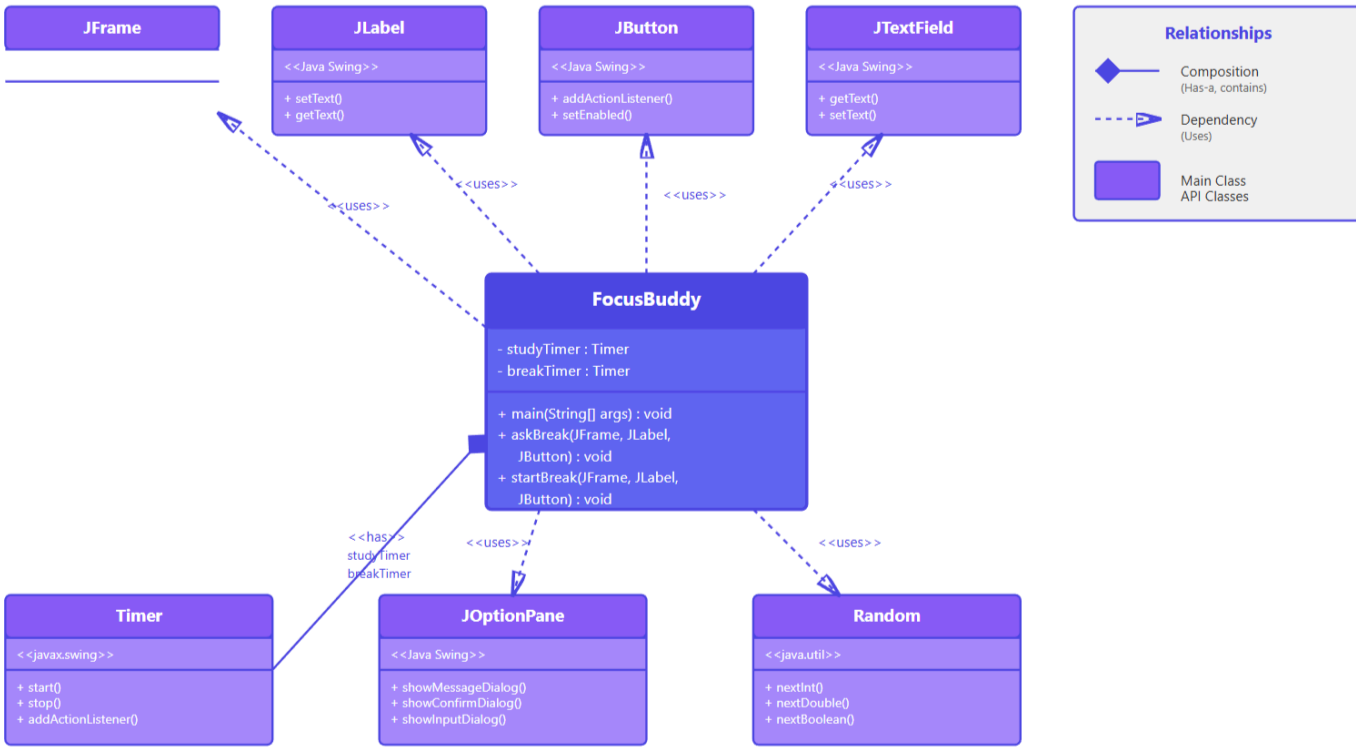


Data Flow:

User inputs time → GUI validates → Event triggers Timer → Timer updates Display → User sees countdown & popups

Timer completion → Motivational quote → Break prompt → Break timer (if accepted) → Session restart

2.2 Class Diagram



Class Description:

FocusBuddy: Main application class that orchestrates the study timer, break timer, and user interactions

Timer (javax.swing.Timer): Provides countdown functionality for both study and break sessions

JFrame, JLabel, JButton, JTextField: GUI components for user interface

JOptionPane: Displays motivational quotes and break confirmation dialogs

Random: Generates random motivational quotes from a predefined collection

CHAPTER -3 COs ATTAINMENT

3.1 CO1 Attainment

CO1 Syllabus	CO1 Concepts Included in Project
Apply fundamental programming constructs such as data types, operators, conditional and iterative statements in Java to develop logic-based solutions for basic computational problems. Students will learn to design simple algorithms, trace execution, and validate logic through hands-on coding tasks.	• Data types • Operators • Conditional logic • Iterative statements • Simple algorithm

3.1.1 Scenarios for CO1 implementation.

1. Data Types (int, String)

Scenario:

The user enters study time in minutes. The program stores this value as an int to perform calculations, and stores motivational quotes in a String array.

2. Operators (*, /, %, --, <=)

Scenario:

The program converts minutes into seconds using *, divides and finds remainder for minutes and seconds using / and %, decreases the timer each second using --, and checks if time is finished using <=.

3. Conditional Statements (if-else)

Scenario:

While the countdown is running, the program checks:

- If the time left becomes 0, it shows a popup with a motivational quote.
- Else, it continues showing the remaining time.
- It also checks whether the user chooses "Yes" for a break.

4. Iterative Logic (Timer Callbacks)

Scenario:

The Swing Timer calls actionPerformed() every 1 second.

Each second, the program updates:

- The remaining minutes and seconds
- The countdown label

This repeated callback works like a loop until the timer ends.

5. Simple Algorithmic Logic (Countdown & Break Timer)

Scenario:

The program follows a simple algorithm:

1. Convert user input to seconds
2. Reduce the time by 1 each second
3. Display updated time
4. When time reaches 0 → show motivational message
5. Ask user if they want a 5-minute break
6. Start break countdown if selected

6. Input Handling (JTextField)

Scenario:

The user types a number (study minutes) into the JTextField.

The program reads this input using getText() and converts it into an integer.

7. Output Display (JLabel, JOptionPane)

Scenario:

The countdown time is displayed live using a JLabel.

When the study session or break ends, the program displays messages and motivational quotes using JOptionPane popup dialogs.

3.1.2 CO1 code screen shot

```
int minutes = Integer.parseInt(minutesField.getText());   // data type + input
int seconds = minutes * 60;                               // operator

studyTimer = new Timer(1000, new ActionListener() {
    int timeLeft = seconds;                               // data type

    public void actionPerformed(ActionEvent e) {

        int m = timeLeft / 60;                            // operator
        int s = timeLeft % 60;                            // operator

        timerLabel.setText("Time left: " + m + "m " + s + "s"); // output

        if (timeLeft <= 0) {                              // conditional
            studyTimer.stop();
            timerLabel.setText("Time's up!");
        }

        timeLeft--;                                       // -- operator
    }
});
```

3.2 CO2 Attainment

CO2 Syllabus	CO2 Concepts Included in Project
Understand and apply 1D & 2D arrays, array operations, searching/sorting techniques, and optimized algorithmic strategies to solve computational problems efficiently.	• Use of 1D array (String[] quotes) • Array creation & memory representation • Array traversal using random index selection • Basic algorithmic logic for selecting and displaying quotes

3.2.1 Scenarios for CO2 implementation

• The project uses a 1D String array to store multiple motivational quotes.
• A random index is selected to display a quote when the timer ends.
• Demonstrates array creation, indexing, and retrieval.

3.2.2 CO2 code screenshot

1. **GUI Component Creation**

```
JFrame frame = new JFrame("Focus Buddy – Study Timer");
frame.setSize(400, 300);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setLayout(new BorderLayout());

JLabel titleLabel = new JLabel("Focus Buddy", SwingConstants.CENTER);
titleLabel.setFont(new Font("Arial", Font.BOLD, 24));

JButton startButton = new JButton("Start Study");
JButton breakButton = new JButton("Start Break");
JButton stopButton = new JButton("Stop");
```

2. **Event Handling**

```
startButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        startStudyTimer();
    }
});

breakButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        startBreakTimer();
    }
});

stopButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        stopTimers();
    }
});
```

3. Timer Logic

```
private static void startStudyTimer() {
    studyTimer = new Timer(1000, new ActionListener() {
        int timeLeft = 1500; // 25 min

        public void actionPerformed(ActionEvent e) {
            if (timeLeft > 0) {
                timeLeft--;
            } else {
                studyTimer.stop();
                JOptionPane.showMessageDialog(null, "Study session complete!");
            }
        }
    });

    studyTimer.start();
}
```

3.3 CO3 Attainment

CO3 Syllabus	CO3 Concepts Included in Project
String handling and immutability. String vs StringBuilder vs StringBuffer. Common string problems. Regular Expressions (regex) -pattern matching and text validation. Bitwise operators. Recursion fundamentals, base & recursive cases, tracing stack frames. Recursive problem-solving.	• String handling & immutability • String manipulation using setText() • Retrieving input as string (getText()) • Converting strings using Integer.parseInt() • Displaying messages with JOptionPane

3.3.1 Scenarios for CO3 implementation

- The project uses strings for input, timer updates, and UI outputs.
- Motivational quotes stored as strings in an array.
- Timer text updated using string concatenation.
- Input is validated through parsing string values.

3.3.2 CO3 code screenshot

1. **User Input as String**

```
int minutes = Integer.parseInt(minutesField.getText());
```

2. **String Array of Quotes**

```
String[] quotes = {
    "A bright start leads to a brilliant finish.",
    "Stay glowing, stay growing.",
    "Every second counts – shine through.",
    "Consistency is golden.",
    "Your hard work lights the way."
};
```

3. **String Concatenation**

```
timerLabel.setText("Time left: " + m + "m " + s + "s");
```

4. **String Message Display**

```
JOptionPane.showMessageDialog(
    frame,
    "Time's up!\n\n" + quote,
    "Study Session Complete",
    JOptionPane.PLAIN_MESSAGE
);
```

3.4 CO4 Attainment

CO4 Syllabus	CO4 Concepts Included in Project
Develop structured and modular programs by applying object-oriented programming principles such as encapsulation, abstraction, and modularization using Java classes, methods, and constructors. Students will transition from procedural to modular design thinking.	• Encapsulation • Abstraction • Modularization

3.4.1 Scenarios for CO4 implementation

1.Abstraction

Definition: Hiding implementation details and exposing only essential features.

Scenario:

- Define an abstract class Shape with an abstract method calculateArea().
- Subclasses like Circle and Rectangle implement the method differently.
- The input doesn't care *how* the area is calculated, only that can call calculateArea()**.**

2.Modularization

Definition: Breaking a large program into smaller, manageable, reusable modules.

Scenario:

- Building a student management system.
- Split it into modules:
    1. StudentModule (handles student data)
    2. CourseModule (handles courses)

3. EnrollmentModule (links students to courses)

- Each module is a separate class/package, making debugging and scaling easier.

3.Constructors

Definition: Special methods used to initialize objects when they are created.

Scenario:

- Every Car object starts with a model name and year.
- Instead of setting them manually after creating the object, you use a constructor.
- In our **FocusBuddy** Java program, there are **NO constructors**.
- So the class uses the default no-argument constructor provided automatically by Java.
- Your class only contains **static methods** (main, askBreak, startBreak).
- We don't create constructors in our code.

4.Encapsulation

- Private fields → hide the internal state (seconds). Public methods → controlled access (start(), reset(), getSeconds()).
- Validation → prevent invalid values (like negative time).

For a Timer, this ensures:

- Nobody can directly set seconds = -10.
- Only safe operations (tick, reset, start) are allowed.
- The timer logic is consistent and reusable.
- tart(), reset(), getSeconds()).
- Validation → prevent invalid values (like negative time).

For a Timer, this ensures:

- Nobody can directly set seconds = -10.
- Only safe operations (tick, reset, start) are allowed.
- The timer logic is consistent and reusable.

3.4.2 CODE SCREEN SHOTS.

1.Class

```java
public class FocusBuddy {
}
```

2.Encapsulation

```
private static Timer studyTimer;
private static Timer breakTimer;
```

```
private static void askBreak(JFrame frame, JLabel timerLabel, JB
    int choice = JOptionPane.showConfirmDialog(
            frame,
            "Would you like to take a 5-minute break?",
```

we used private, which **hides the variables** from outside the class — that *is encapsulation*.

3. Abstraction (Partially — could be improved)

```
JFrame frame = new JFrame("Focus Buddy - Study Timer (Yellow Mode)");
studyTimer = new Timer(1000, new ActionListener() { ... });
```

```
JFrame frame = new JFrame("Focus Buddy – Study Timer (Yellow Mode)");
frame.setSize(420, 320);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setLayout(new FlowLayout());
```

```
JLabel title = new JLabel("Focus Buddy – Study Timer");
title.setFont(new Font("Arial", Font.BOLD, 18));
title.setForeground(Color.BLACK);
```

3. Modularization

```
startButton.addActionListener(e -> {
    try {
        int minutes = Integer.parseInt(minutesField.getText());
        int seconds = minutes * 60;

        startButton.setEnabled(false);
```

4. Method

```
if (choice == JOptionPane.YES_OPTION) {
    startBreak(frame, timerLabel, startButton);
} else {
    JOptionPane.showMessageDialog(frame, "Okay, stay product
    startButton.setEnabled(true);
}
```

3.5 CO5 Attainment

CO5 Syllabus	CO5 Concepts Included in Project
Design extensible and reusable Java programs employing inheritance, polymorphism, abstract classes, interfaces, and reflection API to solve domain-oriented problems with clarity and maintainability. Students will model real-world entities and relationships through effective OOP architecture.	● Interfaces ● Polymorphism ● Reusability Through Modular Functions ● Object/Instance Creation for Domain Objects ● Encapsulation ● Inheritance (Framework-Level) ● Use of Abstract Classes (Framework-Level) ● Domain-Oriented Problem Solving

3.5.1 Scenarios for CO5 implementation

1. **Using Interfaces for Timer Functionality:**

- Study timer ActionListener
- Break timer ActionListener

Each one defines a different action for actionPerformed().
This shows behavioral polymorphism, as the same interface produces multiple behaviors.

2. **Domain Modeling of a Real-World "Study Break System"**

Our FocusBuddy program models:

- A study session
- A timed break
- A motivational notification
- User inputs (minutes)
- Time tracking (countdown)

This reflects turning a real-world activity into software objects.

3. **Reusable Code Through Modular Functions**

Functions like:

- askBreak()

  Handles break confirmation → reusable across any future timer extensions.

- startBreak()

  Starts a break timer → can be reused in multiple modes (Pomodoro mode, long break, short break).

4. **Event-Driven Architecture Using Listeners**

Swing applications respond to user actions (button click) and time events (timer tick).
This is a core implementation of event-driven OOP design.

Our program waits for:

- Button click → starts study timer
- Timer event → updates countdown
- Timer completion → triggers quote and break prompt

This follows the observer/listener pattern, a key OOP concept.

5. **Using Polymorphism to Change Timer Behaviors**

Both study and break timers use:

```
new ActionListener() {

public void actionPerformed(ActionEvent e) { ... }

};
```

- The JVM decides at runtime which actionPerformed() belongs to which timer.

- This demonstrates runtime polymorphism and dynamic dispatch.

6. **Inheritance and Abstract Classes in Swing**

Our program does not manually extend classes, but it uses:

- JFrame
- JButton
- JLabel
- Timer

All these are part of Swing's OOP hierarchy.

7. **Encapsulation for Timer Objects**

By declaring:

private static Timer studyTimer;

private static Timer breakTimer;

- Other classes cannot modify these timers, preserving class integrity.

8. **Random Class for Domain-Based Motivation**

We use Random rand = new Random(); to randomly select motivational quotes.

This adds non-deterministic behavior, making the application dynamic and interactive.

9. GUI Object Composition

JFrame owns several components:

- Title label
- Input field
- Button
- Timer label

This follows object composition, where multiple objects work together to form a meaningful system.

3.5.2.CO5 code screen shots.

1. **Interface**

```
startButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        // button click logic
    }
});
```

2. **Polymorphism**

```
studyTimer = new Timer(1000, new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        // Study timer countdown logic
    }
});

breakTimer = new Timer(1000, new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        // Break timer countdown logic
    }
});
```

3. **Encapsulation**

```
private static Timer studyTimer;
private static Timer breakTimer;
```

4. **Inheritance**

```
JFrame frame = new JFrame("Focus Buddy");
JButton startButton = new JButton("Start Timer");
JLabel timerLabel = new JLabel("Time left: --");
```

5. **Abstract class**

```
// JButton inherits from AbstractButton (an abstract class)
JButton startButton = new JButton("Start Timer");
```

6. **Object creation**

```
JLabel title = new JLabel("Focus Buddy – Study Timer");
JTextField minutesField = new JTextField(10);
Timer studyTimer = new Timer(1000, null);
```

7. **Event-driven oop**

```
startButton.addActionListener(e -> {
    // triggers on click
});
```

8. **Modularity/reusability**

```
private static void askBreak(JFrame frame, JLabel timerLabel, JButton startButton) {
    int choice = JOptionPane.showConfirmDialog(frame,
        "Would you like a 5-minute break?", "Break Time", JOptionPane.YES_NO_OPTION);
}
```

9. Random class

```
String[] quotes = { ... };
Random rand = new Random();
String quote = quotes[rand.nextInt(quotes.length)];
```

3.6 CO6 Attainment

CO6 Syllabus	CO6 Concepts Included in the Project
Implement robust, scalable, and generic Java applications integrating exception handling, file I/O, generics, and collections framework, along with functional programming constructs, to handle real-world data-driven tasks. Students will demonstrate ability to write production-level, fault-tolerant programs.	● Exception Handling ● Java API Classes ● Event-Driven Programming ● Conditional Logic ● Iterative Statements ● Modular & Reusable Methods ● Robust & Fault-Tolerant Behavior ● Simple Algorithms

3.6.1 Scenarios for CO6 implementation.

1. **Exception Handling**

- When the user enters invalid input (like letters instead of numbers), the program catches the error and shows a message:
  "Please enter a valid number."
  This prevents the program from crashing.
- Java API Classes
- The project uses multiple built-in Java classes such as:
  JFrame, JLabel, JButton, JTextField, JOptionPane, Timer, Color, Font, Random
  These APIs help in creating windows, handling timers, generating quotes, and customizing UI.

3. Event-Driven Programming

- When the user clicks the Start Timer button, an event listener runs the timer.
- When the timer ends, another event triggers a popup.
- Clicking "YES" or "NO" in the break dialog triggers different actions.

4. Conditional Logic

- Conditions decide what happens next:
    1. If study time ends → show motivational message.
    2. If user chooses break → start 5-minute timer.
    3. If user clicks "NO" → timer resets and button enables.

5. Iterative Statements

- The countdown timer uses a loop structure inside the Timer to update time every second.
- Break timer also counts down using repetitive execution until it reaches 0.

6. Modular & Reusable Methods

- Separate methods like askBreak() and startBreak() make the code cleaner and more reusable.
- These methods can be used independently without rewriting logic.

7. Robust & Fault-Tolerant Behavior

- The GUI prevents invalid operations by disabling the start button during countdown.
- The program handles all possible user mistakes (invalid input / repeated clicks) without crashing.

8. Simple Algorithms

- Implements a countdown algorithm that converts minutes → seconds, reduces them, and displays remaining time.
- Implements a timed break algorithm for the 5-minute rest period.

3.6.2 CO6 code screen shot

1. **Exception Handling:**

```
try {
    int minutes = Integer.parseInt(minutesField.getText());
} catch (Exception ex) {
    JOptionPane.showMessageDialog(frame, "Please enter a valid number.");
}
```

2. **Java API Classes**

```
JFrame frame = new JFrame("Focus Buddy – Study Timer");
JLabel title = new JLabel("Focus Buddy – Study Timer");
JButton startButton = new JButton("Start Timer");
JTextField minutesField = new JTextField(10);
Timer studyTimer;
Timer breakTimer;
Random rand = new Random();
```

3. **Event-Driven Programming**

-

```
startButton.addActionListener(e -> {
    // start timer logic
});
```

- And timer events

```
studyTimer = new Timer(1000, new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        // countdown logic
    }
});
```

4. **Conditional Logic**

-

```
if (timeLeft <= 0) {
    studyTimer.stop();
    askBreak(frame, timerLabel, startButton);
}
```

- Break Prompt

```
if (choice == JOptionPane.YES_OPTION) {
    startBreak(frame, timerLabel, startButton);
} else {
    startButton.setEnabled(true);
}
```

5. **Iterative Statements (loop behaviour using Timer)**

```
public void actionPerformed(ActionEvent e) {
    timeLeft--;    // repeats every second
}
```

6. **Modular & Reusable Methods**

```
private static void askBreak(JFrame frame, JLabel timerLabel, JButton startButton) { ... }

private static void startBreak(JFrame frame, JLabel timerLabel, JButton startButton) { ... }
```

7. **Robust & Fault-Tolerant Behavior**

```
startButton.setEnabled(false);  // prevents multiple clicks during timer
```

- After break or invalid input

```
startButton.setEnabled(true);
```

8. **Simple Algorithms (Countdown + Break Timer)**

```
int seconds = minutes * 60;

int m = timeLeft / 60;
int s = timeLeft % 60;

timerLabel.setText("Time left: " + m + "m " + s + "s");
```

CHAPTER- 4 FOCUS BUDDY CODE

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

public class FocusBuddy {

    private static Timer studyTimer;
    private static Timer breakTimer;

    public static void main(String[] args) {

        JFrame frame = new JFrame("Focus Buddy – Study Timer (Yellow Mode)");
        frame.setSize(420, 320);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLayout(new FlowLayout());
        frame.getContentPane().setBackground(new Color(255, 230, 120)); // soft yellow bg

        JLabel title = new JLabel("Focus Buddy – Study Timer");
        title.setFont(new Font("Arial", Font.BOLD, 18));
        title.setForeground(Color.BLACK);

        JLabel inputLabel = new JLabel("Enter study duration (in minutes):");
        inputLabel.setForeground(Color.BLACK);

        JTextField minutesField = new JTextField(10);
        minutesField.setBackground(new Color(255, 240, 160)); // lighter yellow
        minutesField.setForeground(Color.BLACK);
        minutesField.setCaretColor(Color.BLACK);

        JButton startButton = new JButton("Start Timer");
        startButton.setBackground(new Color(255, 210, 80)); // darker yellow
        startButton.setForeground(Color.BLACK);
```

```
JLabel timerLabel = new JLabel("Time left: ~m ~s");
timerLabel.setFont(new Font("Arial", Font.BOLD, 16));
timerLabel.setForeground(Color.BLACK);

frame.add(title);
frame.add(inputLabel);
frame.add(minutesField);
frame.add(startButton);
frame.add(timerLabel);

String[] quotes = {
    "A bright start leads to a brilliant finish.",
    "Stay glowing, stay growing.",
    "Every second counts — shine through.",
    "Consistency is golden.",
    "Your hard work lights the way."
};

Random rand = new Random();

startButton.addActionListener(e -> {
    try {
        int minutes = Integer.parseInt(minutesField.getText());
        int seconds = minutes * 60;

        startButton.setEnabled(false);

        studyTimer = new Timer(1000, new ActionListener() {
            int timeLeft = seconds;

            public void actionPerformed(ActionEvent e) {
                int m = timeLeft / 60;
                int s = timeLeft % 60;
```

```
                timerLabel.setText("Time left: " + m + "m " + s + "s");

                if (timeLeft <= 0) {
                    studyTimer.stop();
                    timerLabel.setText("Time's up!");

                    String quote = quotes[rand.nextInt(quotes.length)];
                    JOptionPane.showMessageDialog(
                        frame,
                        "Time's up!\n\n" + quote,
                        "Study Session Complete",
                        JOptionPane.PLAIN_MESSAGE
                    );

                    askBreak(frame, timerLabel, startButton);
                }

                timeLeft--;
            }
        });

        studyTimer.start();

    } catch (Exception ex) {
        JOptionPane.showMessageDialog(frame, "Please enter a valid number.");
    }
});

frame.setVisible(true);
}

private static void askBreak(JFrame frame, JLabel timerLabel, JButton startButton) {
    int choice = JOptionPane.showConfirmDialog(
        frame,
        "Would you like to take a 5-minute break?",
```

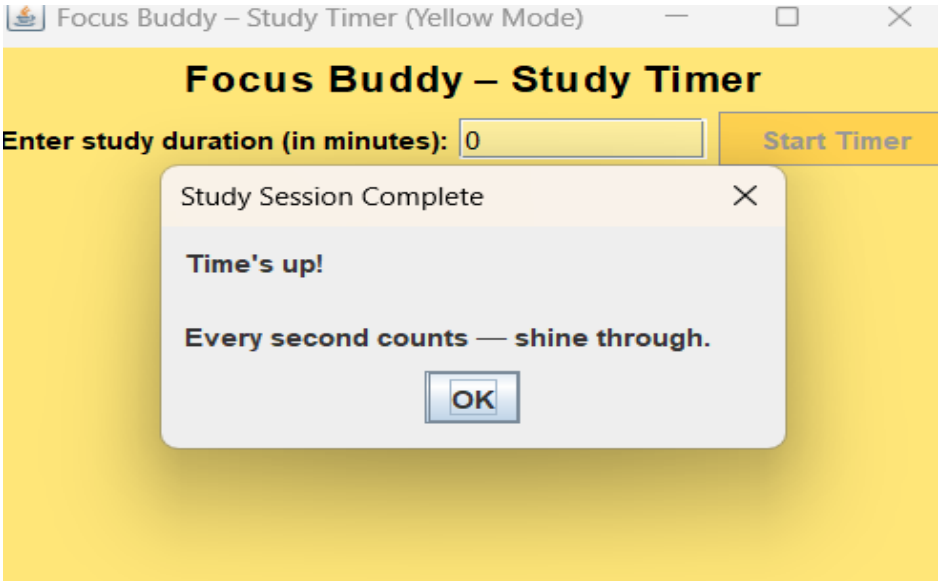
```
            "Break Time",
            JOptionPane.YES_NO_OPTION
        );

        if (choice == JOptionPane.YES_OPTION) {
            startBreak(frame, timerLabel, startButton);
        } else {
            JOptionPane.showMessageDialog(frame, "Okay, stay productive.");
            startButton.setEnabled(true);
        }
    }

    private static void startBreak(JFrame frame, JLabel timerLabel, JButton startButton) {

        breakTimer = new Timer(1000, new ActionListener() {
            int timeLeft = 300;

            public void actionPerformed(ActionEvent e) {
                int m = timeLeft / 60;
                int s = timeLeft % 60;

                timerLabel.setText("Break time left: " + m + "m " + s + "s");

                if (timeLeft <= 0) {
                    breakTimer.stop();
                    JOptionPane.showMessageDialog(frame, "Break over. Time to focus again.");
                    startButton.setEnabled(true);
                }

                timeLeft--;
            }
        });

        breakTimer.start();
    }}
```

CHAPTER -5 TESTING & RESULTS

4.1 Test cases and result

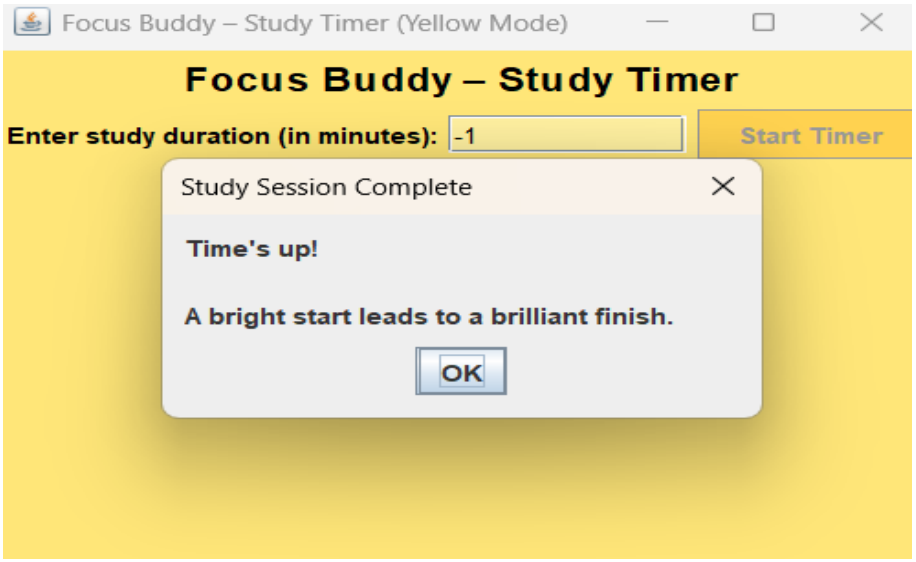
Input:0

Result:



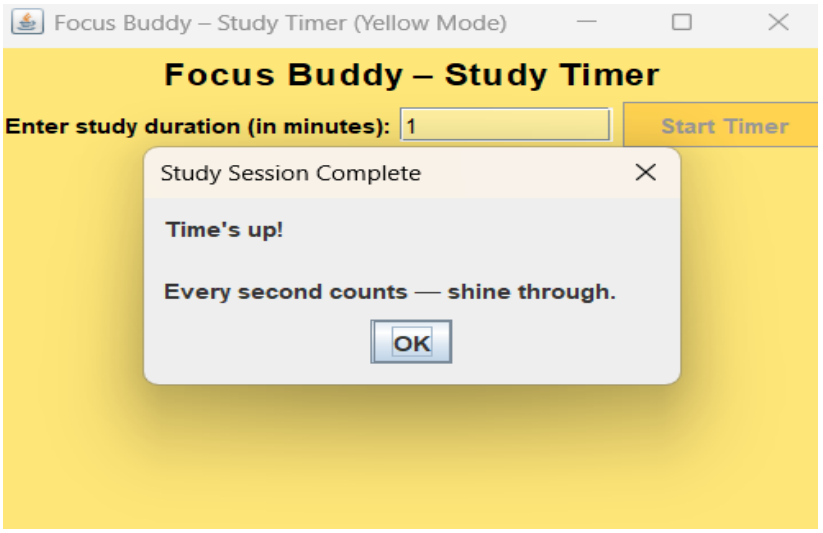
Input:-1

Result:



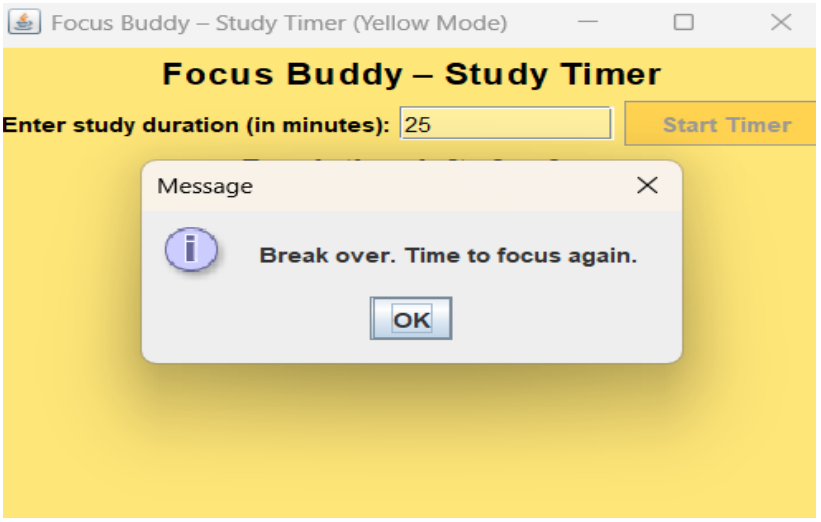
Input:1

Result:



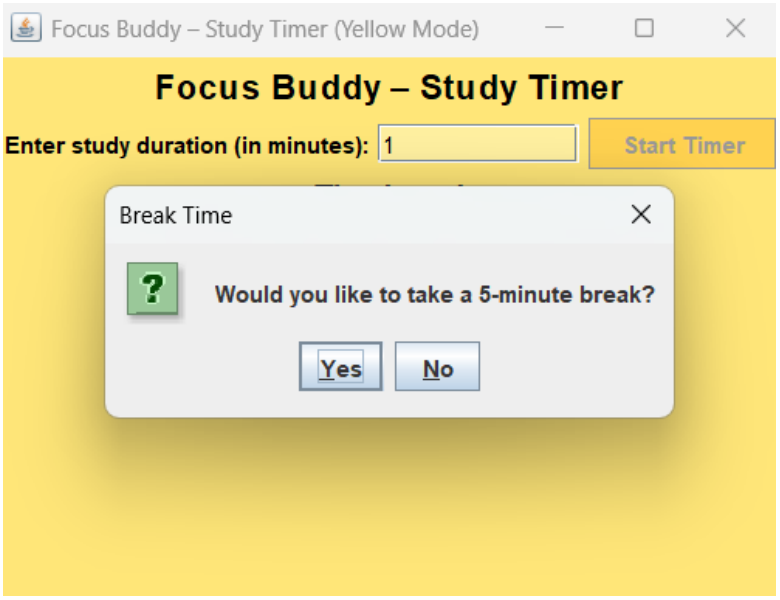
Input:25

Result:



Break: 5

Result:



CHAPTER -6 FUTURE ENHANCEMENTS

6.1 Planned Features

1. **Customizable Study & Break Intervals**

- Allow users to set different durations for study sessions and breaks instead of fixed break time (currently 5 minutes).
- Add options for Pomodoro cycles with long breaks after multiple sessions.

2. **User Profiles & Progress Tracking**

- Enable multiple user profiles with saved preferences.
- Track daily/weekly study hours and present performance statistics with charts or graphs.

3. **Motivational & Personalized Features**

- Provide personalized motivational quotes or tips based on the user's study habits.
- Integrate audio notifications or subtle background music to enhance focus.

4. **Task & Subject Management**

- Allow users to enter specific tasks or subjects for each session.
- Display progress or completion percentage for each task.

5. **Cross-Platform Support**

- Develop mobile or web versions of FocusBuddy for easier access.
- Enable synchronization of sessions across devices for continuity.

6. **Smart Notifications & AI Assistance**

- AI-driven reminders to suggest optimal break times based on user focus patterns.
- Predictive analysis of productivity trends and tips to improve concentration.

7. **Gamification**

- Introduce points, badges, or streaks to motivate consistent study habits.
- Leaderboards for group study or friendly competition.

8. **Enhanced User Interface & Accessibility**

- Themes and color modes for night and day use.
- Accessibility features for visually impaired users or those with learning difficulties.

9. **Integration with Calendar & Tasks Apps**

- Sync study sessions with Google Calendar, Outlook, or task management apps.
- Automated session planning based on upcoming deadlines or exams.

10. **AI Study Assistance**

- Suggest optimized study schedules based on subjects or difficulty levels.
- Provide quick learning resources, flashcards, or quizzes during breaks.

6.2 Possible integrations or optimizations

- **Calendar Integration:** Sync study sessions with Google Calendar or phone reminders.
- **Cloud Sync:** Save progress across devices or platforms.
- **AI-Based Insights:** Provide personalized study tips, productivity predictions, and focus recommendations**.**
- **Performance Optimization:** More efficient timer logic, smoother UI rendering, and better memory handling.
- **Data Visualization Tools:** Graphs using libraries to show weekly/monthly progress.
- **Mobile App Integration:** Extend functionality to Android/iOS for wider accessibility.
- **Better UI Layout:** Panels, icons, animations, or a dashboard-style interface.

CHAPTER -7 CONCLUSION

7.1 Summary of the Project

FocusBuddy is an innovative study timer application developed in Java Swing to help students manage their study sessions effectively and maintain focus. The application allows users to input their desired study duration, after which a countdown timer starts, visually displaying the remaining time. Upon completion, users receive motivational quotes to encourage productivity and maintain a positive mindset.

The application also incorporates a break timer, prompting users to take short rests between study sessions. This promotes the Pomodoro technique, preventing burnout and improving overall concentration. The interface is designed with soft, visually appealing colors to create a calm and motivating environment, making it easier for students to engage with their study routines.

FocusBuddy leverages GUI components, event handling, and timers in Java to create a seamless and interactive user experience. Its primary goal is to enhance time management skills, foster consistent study habits, and balance productivity with rest. The project also lays the foundation for future enhancements, such as user profiles, task tracking, AI-based reminders, gamification, and cross-platform support, which can further increase engagement and efficiency.

7.2 What was achieved

- A fully functional GUI study timer using Java Swing.
- Real-time countdown for both study and break sessions.
- Interactive prompts, alerts, and motivational quotes.
- A clean user interface with themed colors for a calm study environment.
- Enabling and disabling buttons at the right times to prevent incorrect input.
- Reliable event-handling and timer functionality.

7.3 Skills Learned During Development

- **Java Swing Components:** Creating windows, buttons, labels, dialogs, and layouts.
- **Event Handling:** Using ActionListeners to trigger events and respond to user actions.
- **Timer Usage:** Implementing countdown logic with javax.swing.Timer.
- **GUI Design Principles:** Color choices, readability, spacing, and smooth user interaction.
- **Exception Handling:** Validating user inputs and preventing app crashes.
- **Logical Thinking:** Structuring code cleanly, breaking functionality into separate methods, and managing states like start, break, and reset.Software Development Flow: Planning features, solving problems, and designing user-focused tools

CHAPTER -8 REFERENCES

Books

Pressman, R. S., & Maxim, B. R. (2019). *Software Engineering: A Practitioner's Approach* (8th ed.). McGraw- Hill Education.

Freeman, E., & Freeman, E. (2020). *Head First HTML and CSS* (2nd ed.). O'Reilly Media. Duckett, J. (2014). *HTML & CSS: Design and Build Websites*. Duckett, J. (2015). *JavaScript and JQuery: Interactive Front-End Web Development*. Wiley. Sommerville, I. (2016). *Software Engineering* (10th ed.).

Tutorials and Learning Resources

W3Schools. (2025). *HTML, CSS, and JavaScript Tutorials*. Retrieved from https://www.w3schools.com

MDN Web Docs. (2025). *Web Technologies Documentation*. Retrieved from https://developer.mozilla.org

FreeCodeCamp. (2025). *Responsive Web Design Certification*. Retrieved from https://www.freecodecamp.org

GeeksforGeeks. (2025). *Front-end Development and Web Technologies*. Retrieved from https://www.geeksforgeeks.org

Tutorialspoint. (2025). *HTML, CSS, JavaScript, and Web Development Tutorials*. Retrieved from https://www.tutorialspoint.com

APIs and Tools Used

Google Maps Platform. (2025). *Maps JavaScript API Documentation*. Retrieved

from https://developers.google.com/maps

Unsplash API. (2025). *Free High-Resolution Image Access.* Retrieved from https://unsplash.com/developers

OpenWeatherMap. (2025). *Weather API for Travel Assistance.* Retrieved from https://openweathermap.org/api

RapidAPI Hub. (2025). *Travel and Tourism APIs.* Retrieved from https://rapidapi.com/hub

Firebase by Google. (2025). *Authentication and Database Services.* Retrieved from https://firebase.google.com

Documentation and Research References

World Tourism Organization (UNWTO). (2024). *Tourism Data Dashboard.* Retrieved from https://www.unwto.org

Booking.com. (2025). *Accommodation and Destination Insights.* Retrieved from https://www.booking.com

TripAdvisor. (2025). *Tourism and Destination Review Data.* Retrieved from https://www.tripadvisor.com

Travel + Leisure. (2025). *Top Global Destinations and Travel Trends.* Retrieved from https://www.travelandleisure.com

Tourism Review. (2025). *Digital Transformation in the Tourism Industry.* Retrieved from https://www.tourism- review.com

CHAPTER -9 APPENDICES

Abbreviation	Full Form
GUI	Graphical User Interface
OOP	Object-Oriented Programming
API	Application Programming Interface
JVM	Java Virtual Machine
regex	Regular Expression

GITHUB LINK- FocusBuddy

YOUTUBE LINK- FocusBuddy

Installation & Setup Instructions

1. System Requirements

Software:

- Windows OS (7 or higher)
- Java JDK 8+
- IntelliJ / Eclipse (optional for running code)

Hardware:

- 2 GB RAM
- Basic keyboard & mouse
- 100–200 MB free storage

2. Installation Steps

Step 1: Install Java

1. Download JDK (8 or higher) from the official site.
2. Install it and verify using:

```
java -version
```

Step 2: Download the Project

You can do either:

- Download ZIP from GitHub and extract it

  Or
- Clone using Git:

```
git clone <repository-link>
```

Step 3: Open in IDE

1. Open IntelliJ/Eclipse.
2. Go to **File → Open Project**.
3. Select the project folder.

Step 4: Run the Application

1. Open **FocusBuddy.java**.
2. Click **Run**.
3. The GUI window will open automatically.

3. Basic Troubleshooting

- If the window doesn't appear → reinstall JDK.
- If errors show → re-check input or rebuild project.
- If fonts or background look odd → update Java GUI libraries (Swing is required).

User Manual / Guide

1. Starting the Application

- Launch the project from your IDE.
- A window will appear with the timer interface.

2. Using the Study Timer

1. Enter study time in minutes in the textbox.
2. Click **Start Timer**.
3. Timer begins counting down automatically.

3. After the Timer Ends

- A popup message appears with **"Time's up!"**
- A random motivational quote is shown.

4. Break Timer Feature

- You will be asked if you want a **5-minute break**.
- If **Yes**, break timer starts and counts down.
- If **No**, you can start another session anytime.

5. Error Handling

- If user enters invalid input (letters or empty field), a popup shows:
  "Please enter a valid number."

6. Closing the App

- Click the **X** in the top right corner.

- All timers will stop automatically.

Geo Tag Photos