

VIVEKANANDA GLOBAL UNIVERSITY

Master of Computer Applications

# Object Oriented Programming Using JAVA

(PGCSA111)

Project Title: Comprehensive Email Validator

# Project Report

PROJECT GUIDE: SUBMITTED BY:

Mr. Narayan Vyas

|  |
| --- |
| VANSHITA JAIN: 24CSA3BC062  SUMIT KUMAR BAIRWA: 24CSA3BC107  SUNITA MAURYA: 24CSA3BC028 |
|  |
|  |

# ACKNOWLEDEGEMENT

I have taken this opportunity to express my gratitude and humble regards to the Vivekananda Global University to provide an opportunity to present a project on the “Comprehensive Email Validator”.

I would also be thankful to my project guide Mr. Narayan Vyas to help me in the completion of my project and the documentation. I have taken efforts in this project, but the success of this project would not be possible without their support and encouragement.

I would like to thanks our dean sir “Dr. R C Tripathi” to help us in providing all the necessary books and other stuffs as and when required. I show my gratitude to the authors whose books has been proved as the guide in the completion of my project I am also thankful to my classmates and friends who have encouraged me in the course of completion of the project.

Thanks

Place: Jaipur Date: 29-03-2025

# DECLARATION

We hereby declare that this Project Report titled “Comprehensive Email Validator” submitted by us and approved by our project guide, to the Vivekananda Global University, Jaipur is a bonafide work undertaken by us and it is not submitted to any other University or Institution for the award of any degree diploma / certificate or published any time before.

Project Guide: Mr. Narayan Vyas

Student Name:

VANSHITA JAIN: 24CSA3BC062

SUMIT KUMAR BAIRWA: 24CSA3BC107

SUNITA MAURYA: 24CSA3BC028

# Table of Contents

1. [Abstract](#1-abstract)
2. [Introduction](#2-introduction)
3. [Objectives](#3-objectives)
4. [System Requirements](#4-system-requirements)  
     4.1 [Hardware Requirements](#4-system-requirements)  
     4.2 [Software Requirements](#4-system-requirements)
5. [System Design](#5-system-design)
6. [Implementation Details](#6-implementation-details)  
     6.1 [Classes Used](#6-implementation-details)  
     6.2 [Key Features](#6-implementation-details)
7. [Code Explanation](#7-code-explanation)  
     7.1 [Validation Logic](#7-code-explanation)  
     7.2 [Feedback Mechanism](#7-code-explanation)
8. [Sample Execution](#8-sample-execution)
9. [Advantages & Disadvantages](#9-advantages--disadvantages)  
     9.1 [Advantages](#9-advantages--disadvantages)  
     9.2 [Disadvantages](#9-advantages--disadvantages)
10. [Future Enhancements](#10-future-enhancements)
11. [Conclusion](#11-conclusion)
12. [References](#12-references)

**Comprehensive Email Validator** **Results in Java**

## Abstract:

The **Comprehensive Email Validator** is a **Java-based** system designed to verify email addresses by checking their format against predefined **regular expressions (regex)**.

Key Points:

### Java-Based System:

* The validator is written in Java, making it platform-independent and reliable.
* It can be integrated into various applications, including web platforms, desktop software, and command-line tools.

### Regex for Email Validation:

* Regular expressions are patterns used to match text.
* The system uses regex to ensure that email addresses conform to standard formats, such as example@domain.com.
* It checks for necessary components like the username, @ symbol, domain name, and top-level domain (TLD).

### Compliance with Email Standards:

* The validator follows **RFC 5322**, the official specification for email formatting.
* It prevents common formatting errors, such as missing domain names, special characters in the wrong places, or excessive dots (..) in email addresses.

### Detailed Feedback for Invalid Emails:

If an email is **invalid**, the system provides specific reasons, such as:

* **"Missing '@' symbol"**
* **"Invalid domain format"**
* **"TLD missing or incorrect"**

This feedback helps users correct errors efficiently.

## Introduction:

Email validation plays a critical role in ensuring **authentication and data integrity** in various applications, such as user registrations, form submissions, and account verifications. A valid email address is essential for establishing communication with users and preventing issues like fake or incorrect email entries.

This **Comprehensive Email Validator** is designed to accurately validate email addresses using **regular expressions (regex)**. The system scans an input email for proper syntax, detects **invalid patterns**, and provides **detailed feedback** if an email is incorrectly formatted.

By ensuring that users enter properly structured email addresses, the system helps to:

* Prevent **spam registrations** and **fake accounts**.
* Improve **data quality** in databases.
* Reduce **bounced emails** due to incorrect formatting.
* Enhance **user experience** by offering clear error messages for invalid inputs.

## Objectives:

The primary goal of this project is to create an **efficient and reliable email validation system**. The specific objectives include:

### Develop a reliable email validator using Java:

* + Implement an email validation system using **Java**, ensuring portability and efficiency.
  + Design the system to be lightweight and easy to integrate into larger applications.

### Implement regex-based validation:

* + Utilize **regular expressions (regex)** to enforce **proper email format standards**.
  + Ensure compliance with industry standards, such as **RFC 5322**.

### Provide detailed feedback on invalid emails:

* + Detect specific formatting issues, such as **missing '@' symbols, invalid domains, or incorrect characters**.
  + Display clear error messages to help users correct their input.

### Create a simple command-line interface (CLI):

* + Implement a user-friendly command-line tool to allow users to validate email addresses efficiently.
  + Provide a straightforward way to test and debug email inputs without requiring a graphical interface.

By achieving these objectives, the system will offer a **robust email validation mechanism** that enhances **user experience, prevents errors, and improves data quality** in various applications.

## System Requirements:

The system requirements define the necessary **hardware and software** needed to run the Comprehensive Email Validator efficiently.

**Hardware Requirements:**

These specifications ensure smooth execution of the Java-based email validator:

* **Intel Core i3 or higher** → Provides sufficient processing power for running Java applications.
* **4GB RAM** → Ensures smooth execution of the program, even alongside other applications.
* **100MB storage** → The program requires minimal space for installation and execution.
* **Windows/Linux/MacOS** → The system is cross-platform and can run on any operating system that supports Java.

**Software Requirements:**

The following software components are necessary for running and developing the validator:

* **JDK 8+ (Java Development Kit)** → Required to compile and run Java programs. Using **JDK 8 or higher** ensures compatibility with modern Java features.
* **IDE (IntelliJ IDEA, Eclipse, or other Java IDEs)** → A development environment for writing and debugging the Java code.
* **JRE (Java Runtime Environment)** → Necessary to execute Java programs on any system.

These system requirements allow developers and users to run the validator efficiently across various platforms.

## System Design:

The Comprehensive Email Validator follows a simple yet effective **workflow** to validate email addresses. The design consists of three key steps:

1. **User Input:**
   * The user provides an email address as input through the **command-line interface (CLI)**.
2. **Validation Process (Regex-Based Check):**
   * The system applies **regular expressions (regex)** to analyze the email format.
   * The validation process checks for:
     + The presence of the @ symbol.
     + A valid domain name (e.g., gmail.com).
     + Proper placement of special characters (e.g., no spaces, no consecutive dots).
     + A valid top-level domain (TLD) (e.g., .com, .org).
3. **Feedback and Output:**
   * If the email is **valid**, the system prints a confirmation message:

example@domain.com is VALID.

* + If the email is **invalid**, the system provides a reason for rejection:

invalid@com is NOT valid. Reason: Missing domain.

This **simple yet effective design** ensures accurate **email format validation** while offering **real-time feedback**, making it user-friendly and efficient for various applications.

## Implementation Details:

### Classes Used:

### The implementation is based on a single Java class:

### ComprehensiveEmailValidator: This class contains the logic for validating email addresses using regular expressions (regex).

### Key Features:

* + **Regex-Based Validation** → Uses regex patterns to check if an email is correctly formatted.
  + **Real-Time Feedback** → Instantly provides reasons if an email is invalid.
  + **Command-Line Interface (CLI)** → The program is lightweight and runs efficiently on the command line.

## Code Explanation:

### Package Declaration:

The line 'package main.java;' declares the package in which the class resides. It helps organize the classes in a structured hierarchy.

### Imports

- 'Scanner' is used for reading user input from the command line.  
- 'Pattern' is used for compiling and applying regex patterns for email validation.

### Class Declaration

The class 'ComprehensiveEmailValidator' encapsulates the logic for validating emails using regex and giving meaningful feedback to the user.

### Regex Definition

EMAIL\_REGEX defines the pattern for a valid email address. It checks for proper formatting, disallows invalid characters, and ensures the email doesn't start/end with incorrect symbols.  
EMAIL\_PATTERN compiles this regex for reuse.

### Email Validation Method

'isValidEmail(String email)' checks if the input email matches the defined regex pattern. Returns true if valid, false otherwise.

### Invalid Reason Feedback

'getInvalidReason(String email)' explains why an email might be invalid, such as missing '@', starting/ending with dots, or having consecutive dots.

### Main Method

This is the entry point. It runs in a loop to repeatedly take user input and validate emails.

* If the email is valid, it prints a confirmation message.
* If invalid, it uses getInvalidReason() to explain the error.
* The loop exits if the user types 'exit' or 'quit'.

### Sample Execution Flow

* Input: example@domain.com  
  Output: example@domain.com is a VALID email address.
* Input: invalid@com  
  Output: invalid@com is NOT a valid email address. Reason: Email format is incorrect.

## Sample Execution:

## 

## Advantages and Disadvantages:



### Advantages:

* + - **Detailed Feedback** → Provides reasons for invalid emails.
    - **Accurate Regex Validation** → Ensures emails follow industry-standard formats.
    - **Prevents Incorrect Email Entries** → Reduces fake or incorrectly formatted emails in databases.

### Disadvantages:

* **No Domain Existence Verification** → Only checks email format, not whether the domain actually exists.
* **Limited to Command-Line Use** → No graphical interface (GUI) for non-technical users.

## Conclusion:

The **Comprehensive Email Validator** efficiently validates email formats using **regular expressions**, ensuring that user-provided email addresses are correctly structured. Its **detailed feedback mechanism** improves **data integrity** in applications requiring user authentication.

Future improvements, such as **domain verification and GUI development**, will enhance the validator's functionality and usability across different platforms.

## References:

* Java Documentation: <https://docs.oracle.com/javase/>
* Java Collections Framework: <https://www.geeksforgeeks.org/collections-in-java/>
* Object-Oriented Programming in Java:<https://www.w3schools.com/java/java_oop.asp>