KONGU ENGINEERING COLLEGE

(Autonomous)

Perundurai, Erode – 638 060

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

Currency Converter System

AN MICRO PROJECT REPORT

for

JAVA PROGRAMMING (22ITC31)

Submitted by

D.LOKHASELVAN (23EIR055)

K.MADHAVAN (23EIR056)

A.S.PAVINSAKTHI (23EIL121)

KONGU ENGINEERING COLLEGE

(Autonomous)

Perundurai, Erode – 638 060

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

Estd : 1984

Transform Yourself

BONAFIDE CERTIFICATE

Name : **D. LOKHASELVAN (23EIR055)**

K. MADHAVAN (23EIR056)

A.S. PAVINSAKTHI (23EIL121)

Course Code : **22ITC31**

Course Name : **JAVA PROGRAMMING**

Semester : **III**

Certified that this is a bonafide record of work for application project done by the above students for **22ITC31 – JAVA PROGRAMMING** during the academic year **2024 - 2025.**

Submitted for the Viva Voce Examination held on _____

Faculty In-Charge **Head of the Department**

INDEX

ABSTRACT

In a globalized economy, currency conversion has become a vital necessity for individuals and businesses engaging in international transactions. Despite the availability of various currency converter tools, many existing systems face challenges related to accuracy, user experience, and real-time data updates. Exchange rates fluctuate constantly due to market dynamics, economic policies, and geopolitical events, creating a need for an accessible and reliable solution that can provide users with accurate and up-to-date conversion rates. Additionally, many current solutions either lack essential features such as multi-currency support, historical data analysis, and offline functionality, or are too complex for everyday users, leaving a gap in the market for a user-friendly yet robust currency converter system.

To address these issues, the proposed currency converter system leverages modern software development methodologies and real-time data integration. The system utilizes API connections with reliable financial data providers to fetch live exchange rates, ensuring the accuracy of conversions. The user interface is designed with simplicity and ease of use in mind, allowing users to input amounts, select currencies, and view conversion results instantly. Advanced features such as historical exchange rate analysis, currency trend charts, and offline access are incorporated to provide added value.

The implemented currency converter system successfully addresses the identified gaps, providing an intuitive and efficient tool for both casual users and financial professionals. Preliminary testing indicates that the system delivers accurate and up-to-date conversions, improving user experience by offering a comprehensive set of features. The integration of real-time data ensures minimal discrepancies, while the inclusion of historical data and offline functionality caters to a broader audience. The results suggest that the system can effectively meet the needs of various user groups, enhancing decision-making in international financial transactions and providing a reliable resource for currency conversion.

PROBLEM STATEMENT

Background of the Problem

The rapid pace of globalization has significantly increased the volume of international trade, travel, and financial transactions. Individuals and businesses now interact with foreign currencies more frequently than ever before, making currency conversion a routine necessity. Whether for personal use, such as tourists converting local currencies while traveling, or for professional use, like businesses settling international invoices, accurate currency conversion is critical. However, the process of converting one currency to another involves complexities due to constantly fluctuating exchange rates, economic policies, and varying conversion fees across different platforms.

Historically, currency conversion was a manual process handled by banks and currency exchange services. Exchange rates were set by financial institutions based on market conditions and were updated at regular intervals. Users often had to rely on published exchange rates in newspapers or visit physical exchange bureaus to obtain the rates. This process was time-consuming, expensive, and prone to errors, especially with significant differences between buy and sell rates imposed by exchange services.

In recent years, technological advancements have given rise to online currency converters, mobile apps, and software that offer more convenient and immediate access to exchange rates. Despite these developments, several issues persist. Existing tools often lack real-time updates, have limited currency support, and vary in accuracy, causing discrepancies in financial transactions. Furthermore, many of these solutions are designed for specific user groups, such as financial professionals or casual users, leading to a fragmented market where no single tool effectively meets the diverse needs of all users.

Past and Present Status of the Problem

The traditional methods of currency conversion were cumbersome and error-prone, especially when exchange rates changed rapidly due to market volatility. In the pre-digital era, users often experienced significant discrepancies between the rates offered by different providers. The lack of transparency in pricing and the added fees charged by banks or exchange services made it difficult for users to accurately estimate the costs of currency conversion. This situation often resulted in financial losses for travelers, businesses, and investors who needed to exchange large sums of money.

With the advent of the internet and mobile technology, digital currency converter tools emerged, promising greater convenience and accuracy. Early solutions were basic, providing only a simple exchange rate lookup without real-time data integration. These tools depended on daily updates from central banks, which did not reflect rapid market changes. As a result, users could not rely on these tools for accurate conversions during periods of economic instability or sudden currency fluctuations.

In the current digital age, online currency converters and mobile applications have become more sophisticated, offering features like real-time exchange rates, historical data analysis, and support for multiple currencies. Services such as Google Currency Converter, XE, and OANDA have become popular, providing easy access to exchange rates. However, despite these improvements, significant challenges remain. Many platforms still face issues with data accuracy, as rates may not update frequently enough to reflect real-time market conditions. Additionally, discrepancies between different providers can create confusion for users who rely on multiple sources for exchange rates.

Security is another concern in the present landscape. As more users turn to online platforms for currency conversion, the risk of data breaches and unauthorized access to personal financial information has increased. Many users are hesitant to trust online services due to fears of privacy violations, fraud, and the misuse of sensitive information. These concerns highlight the need for secure, reliable, and accurate

currency conversion systems that can cater to a diverse user base while addressing the complexities of exchange rate fluctuations and data privacy.

Existing Solutions and Gaps

Existing currency converter tools have evolved to offer a range of features, from basic exchange rate lookups to advanced financial analytics. Popular solutions like XE Currency, Google Currency Converter, and financial platforms such as Bloomberg and Reuters provide extensive currency conversion capabilities. Mobile applications have made it easier for travelers and businesses to access exchange rates on the go, enhancing convenience and usability. These tools typically integrate with financial data providers and use APIs to fetch current exchange rates, offering a more dynamic approach than traditional methods.

Despite the availability of various solutions, significant gaps remain. Many existing tools are limited in scope, focusing on either simple conversion without additional insights or providing complex financial data that is not accessible to the average user. For instance, basic currency converter apps may offer real-time rates but lack features like offline access, historical data, and rate alerts. On the other hand, professional financial software can be overly complex and intimidating for casual users who do not require detailed market analysis.

Furthermore, there is often a lack of consistency in the data provided by different platforms. Exchange rates may vary across different services due to the use of different data sources or update frequencies. This inconsistency can lead to confusion and inaccuracies in financial transactions, especially when dealing with large sums or during periods of high market volatility. Additionally, issues such as hidden fees, transaction costs, and unfavorable rates offered by certain platforms can negatively impact the user experience.

METHODOLOGY

The methodology for developing a currency converter system involves various stages, including requirement analysis, design, implementation, testing, and deployment. The primary objective is to create an accurate, real-time, and user-friendly tool that can effectively handle currency conversion for different users. This methodology explores several methods, data sources, and technologies used to achieve the desired outcome while evaluating their advantages and disadvantages.

2. Data Collection and Exchange Rate Sources

The core of a currency converter system is the accuracy and reliability of exchange rate data. The following methods are used for data collection:

2.1 API Integration with Financial Data Providers

Most modern currency converters rely on Application Programming Interfaces (APIs) from trusted financial data providers such as Open Exchange Rates , CurrencyLayer , Alpha Vantage , and Forex . These APIs offer real-time exchange rate data, historical rates, and additional financial metrics.

Pros:

Real-time Updates:APIs provide up-to-the-minute exchange rates, reducing discrepancies during currency conversions.

Wide Currency Support: Most APIs support numerous global currencies, catering to users from different countries.

Ease of Integration: APIs are relatively easy to integrate into web and mobile applications, allowing seamless data access.

Cons:

Subscription Costs: High-quality APIs often require paid subscriptions, which may be a barrier for small-scale projects.

Rate Limits: Free or basic versions of APIs typically impose rate limits, restricting the number of requests made per day.

Dependence on Third Parties: The system's accuracy and reliability heavily depend on the data provider's service availability.

2.2 Web Scraping

An alternative to using APIs is web scraping, where exchange rate data is extracted directly from trusted websites such as Google Finance , XE , or OANDA . Web scraping involves using scripts to collect data at specified intervals.

Pros:

Cost-Effective: It eliminates the need for paid API subscriptions, making it a budget-friendly solution for smaller projects.

Custom Data Collection: Developers can extract specific data points based on their requirements.

Cons:

Legal and Ethical Issues: Web scraping may violate the terms of service of the source website.

Data Inconsistency: Websites may update their structure, causing scraping scripts to fail and resulting in inaccurate data.

Performance Overhead: Scraping requires frequent data fetching, which can be resource-intensive and slow.

3. System Design and Architecture

The architecture of the currency converter system is designed to be modular, scalable, and secure. It generally includes the following components:

3.1 Client-Side Application (Frontend)

The frontend is responsible for user interaction and input. It is designed using web technologies such as HTML , CSS , and JavaScript , or through mobile frameworks like React Native or Flutter for mobile apps.

Pros:

- User-Friendly Interface: A well-designed frontend enhances user experience by providing intuitive navigation and a clean layout.

- Cross-Platform Support: Using responsive frameworks allows the system to work seamlessly on desktops, tablets, and smartphones.

Cons:

- Complexity in Design: Creating an interface that is both simple for casual users and informative for professionals can be challenging.

- Dependency on Network: The client-side application requires internet access for real-time data fetching, limiting offline use.

3.2 Backend Server

The backend server processes user requests, fetches real-time exchange rates from APIs or web scraping, and performs the conversion calculations. Popular backend technologies include Node.js , Django , and Flask .

Pros:

- Centralized Data Processing: A backend server centralizes data management, ensuring consistent conversion results for all users.

- Scalability: Backend frameworks are scalable and can handle increased traffic as the user base grows.

Cons:

- Maintenance Complexity: Regular updates and maintenance are required to ensure the backend remains secure and efficient.

- Latency Issues: Depending on server location and network speed, there may be delays in data retrieval and response times.

3.3 Database Management

For storing historical exchange rates, user preferences, and transaction records, a database system like MySQL , PostgreSQL , or MongoDB is used.

Pros:

- Data Persistence: Databases enable long-term storage of historical data, which can be useful for analysis and reporting.

- Efficient Querying: SQL-based databases allow efficient data querying and manipulation.

Cons:

- Security Risks: Databases can be vulnerable to SQL injection attacks if not properly secured.

- Data Synchronization: Ensuring that exchange rates are regularly updated in the database can be challenging during high volatility.

4. Implementation Methods

4.1 Real-Time Currency Conversion Algorithm

The core algorithm for currency conversion involves fetching the latest exchange rate and calculating the equivalent value based on user input. The formula used is:

$$ \text{Converted Amount} = \text{Input Amount} \times \text{Exchange Rate} $$

Pros:

- Simplicity: The formula is straightforward, making it easy to implement in any programming language.

- Accuracy: Real-time data fetching ensures that conversions are based on the most recent rates.

Cons:

- Dependence on Data Source: The accuracy of the conversion is entirely dependent on the reliability of the exchange rate data source.

4.2 Caching for Offline Access

To address the issue of network dependency, the system can implement a caching mechanism. It stores the most recent exchange rates locally, allowing users to perform conversions even without an internet connection.

Pros:

- Improved Accessibility: Users can access basic conversion features even when offline.

- Reduced Load on APIs: Caching reduces the number of API requests, minimizing costs and avoiding rate limits.

Cons:

- Outdated Rates: Cached data may become outdated, especially in volatile market conditions, leading to inaccurate conversions.

- Storage Limitations: Storing large amounts of data locally may not be feasible on devices with limited storage.

5. Testing and Evaluation

5.1 Unit Testing

Unit testing involves testing individual components of the system, such as the currency conversion algorithm, API integration, and database queries, to ensure they function correctly.

Pros:

- Early Error Detection: Identifying issues early in the development process reduces the risk of critical failures later.

- Improved Code Quality: Unit testing helps maintain high code standards and reliability.

Cons:

- Time-Consuming: Writing extensive unit tests can be time-consuming and may delay the development process.

- Limited Scope: Unit tests do not cover the entire system's functionality, requiring additional testing methods.

5.2 User Acceptance Testing (UAT)

User Acceptance Testing involves real users testing the application to identify usability issues, bugs, and feedback for improvements.

Pros:

- Real-World Feedback: UAT provides insights into how users interact with the system and reveals issues not identified during development.

- Enhanced User Experience: Incorporating user feedback helps tailor the system to meet user needs better.

Cons:

- Subjective Results: User feedback may vary, making it challenging to standardize improvements.

- Time and Resource Intensive: UAT requires significant time and coordination with users.

The methodology for developing a currency converter system involves a comprehensive approach, utilizing real-time data integration, effective system design, robust algorithms, and rigorous testing. Each method has its strengths and limitations, but together they provide a framework for creating an accurate, reliable, and user-friendly currency converter. By addressing the challenges identified during development, the system can serve as a valuable tool for various users, enhancing financial decision-making in an interconnected global economy.

IMPLEMENTATION

The implementation of the **Currency Converter** involves creating a system that allows users to convert between different currencies using predefined exchange rates. The core objective is to provide a user-friendly application that performs accurate currency conversions, manages a list of countries with their exchange rates, and offers various features like updating, deleting, and inserting new countries into the system. The solution is implemented using Java, focusing on simplicity, modular design, and ease of use.

Setting Up the Project

To begin with the implementation:

- **Development Environment**: We use an Integrated Development Environment (IDE) like **Eclipse** or **IntelliJ IDEA** to write and manage the Java code.
- **Java Version**: Ensure that **JDK 8 or later** is installed on your system to compile and run the program.
- **Project Structure**: We create a Java project with a single file named CurrencyConverter.java to contain the main program logic.

Design of the System

The currency converter system is built with a straightforward design using **ArrayLists** for storing countries and their corresponding exchange rates. This choice is due to the dynamic nature of ArrayLists, which allows us to easily add, remove, or update elements.

Key Components:

1. **Data Storage**: ArrayList<String> countries and ArrayList<Double> exchangeRates store country names and their exchange rates respectively.
2. **Methods**: The program includes several methods for handling currency conversion and data manipulation:

- convert()
- addCountry()
- updateRate()
- deleteCountry()
- insertCountry()

3. **User Interaction**: A menu-driven interface facilitates user input and navigation through various options.

4. **Implementation Details**
5. **4.1 Data Storage and Initialization**
6. We begin by defining two **ArrayLists**: countries for storing country names and exchangeRates for their respective exchange rates. These lists are initialized in the **CurrencyConverter** constructor.

```
private ArrayList<String> countries;

private ArrayList<Double> exchangeRates;


// Constructor

public CurrencyConverter() {

    countries = new ArrayList<>();

    exchangeRates = new ArrayList<>();

}
```

Pre-Filled Data:

- We add initial data for common currencies like the USA (USD), UK (GBP), Euro (EUR), Japan (JPY), and Australia (AUD) to allow immediate testing and conversion without requiring user input.

```
converter.addCountry("USA", 0.012);

converter.addCountry("UK", 0.010);

converter.addCountry("EURO", 0.011);

converter.addCountry("Japan", 1.45);

converter.addCountry("Australia", 0.018);
```

Currency Conversion Logic

The **convert** method is central to the application. It takes three inputs: the source country (from), the destination country (to), and the amount in the source currency. The method retrieves the exchange rates for both countries and performs the conversion using the formula:

Converted Amount=(fromRateamount)×toRate

```
public double convert(String from, String to, double amount) {

    int fromIndex = countries.indexOf(from);

    int toIndex = countries.indexOf(to);


    if (fromIndex == -1 || toIndex == -1) {

        System.out.println("Invalid country.");

        return -1;

    }


    double fromRate = exchangeRates.get(fromIndex);

    double toRate = exchangeRates.get(toIndex);
```

```
    return (amount / fromRate) * toRate;
}
```

Error Handling:

- If the user inputs an invalid country name (not found in the list), the method prints an error message and returns -1, indicating failure.

Adding and Managing Countries

The application allows users to add new countries, update exchange rates, delete existing entries, and insert countries at specific positions:

Add Country:

- The addCountry method appends a new country and its exchange rate to the end of the list.

```
public void addCountry(String name, double rate) {
    countries.add(name);
    exchangeRates.add(rate);
    System.out.println("Country added successfully.");
}
```

Update Exchange Rate:

- The updateRate method updates the exchange rate for an existing country. It locates the country by its index and modifies its exchange rate.

```
public void updateRate(String country, double newRate) {
    int index = countries.indexOf(country);
    if (index != -1) {
```

```
        exchangeRates.set(index, newRate);
        System.out.println("Exchange rate updated successfully.");
    } else {
        System.out.println("Country not found.");
    }
}
```

Delete Country:

- The deleteCountry method removes a country and its corresponding exchange rate from both lists.

```
public void deleteCountry(String country) {
    int index = countries.indexOf(country);
    if (index != -1) {
        countries.remove(index);
        exchangeRates.remove(index);
        System.out.println("Country deleted successfully.");
    } else {
        System.out.println("Country not found.");
    }
}
```

Insert Country:

- The insertCountry method allows users to insert a new country at a specified position.

```
public void insertCountry(int position, String name, double rate) {
    if (position >= 0 && position <= countries.size()) {
        countries.add(position, name);
        exchangeRates.add(position, rate);
        System.out.println("Country inserted successfully.");
    } else {
        System.out.println("Invalid position.");
    }
}
```

User Interface and Input Handling

The **main** method contains a loop that presents a menu of options to the user. It continuously prompts the user for input until they choose to exit.

```
while (true) {
    System.out.println("\n1. Convert Currency");
    System.out.println("2. Add Country");
    System.out.println("3. Update Exchange Rate");
    System.out.println("4. Delete Country");
    System.out.println("5. Insert Country");
    System.out.println("6. Exit");
    System.out.print("Choose an option: ");
    int choice = scanner.nextInt();
```

```
switch (choice) {
    case 1: // Convert Currency
        // User inputs for conversion
        break;
    case 2: // Add Country
        // Adding new country
        break;
    case 3: // Update Rate
        // Updating exchange rate
        break;
    case 4: // Delete Country
        // Deleting a country
        break;
    case 5: // Insert Country
        // Inserting a country
        break;
    case 6: // Exit
        scanner.close();
        return;
    default:
```

```
            System.out.println("Invalid choice. Try again.");
        }
    }
```

Testing and Validation

- **Unit Testing**: We test individual methods such as convert(), addCountry(), and updateRate() to ensure they handle various inputs correctly.
- **User Acceptance Testing**: We test the entire program with different inputs to validate user experience and functional accuracy.
- The implementation of the **Currency Converter** program is straightforward, focusing on core features such as currency conversion and country management. By using Java's ArrayList for dynamic data storage, we achieve efficient data manipulation. The console-based menu allows users to interact with the program easily. Future enhancements could include integrating real-time exchange rate APIs and developing a graphical interface to make the application more versatile and user-friendly.
- This comprehensive explanation should provide a clear understanding of the proposed method and how the problem statement is addressed through this implementation.

RESULT AND DISCUSSTION

The results of the proposed **Currency Converter** program demonstrate the basic functionality of converting currency, managing exchange rates, and handling user inputs in a straightforward manner. This section discusses the effectiveness of the implemented solution, evaluates its performance, and compares it with existing currency converter solutions. Additionally, potential improvements and future directions are highlighted.

Results of the Proposed Implementation

The implemented currency converter program successfully performs the following tasks:

2.1 Accurate Currency Conversion

The core feature of the system, currency conversion, operates as expected. Users can input the amount they wish to convert, along with the source and destination currencies. The program accurately retrieves exchange rates from its stored list and calculates the equivalent value in the desired currency using the formula:

Converted Amount=(amountfromRate)×toRate\text{Converted Amount} = \left(\frac{\text{amount}}{\text{fromRate}}\right) \times \text{toRate}Converted Amount=(fromRateamount)×toRate

Test Case Example:

- **Input**: Convert 1000 INR from **USD** to **JPY**
- **Exchange Rates**:
    - INR to USD: 0.012
    - INR to JPY: 1.45

    Output:

    Converted Amount=(0.0121000)×1.45=120833.33 JPY

The output confirms the expected calculation, indicating that the program correctly handles currency conversion based on given exchange rates.

2.2 User-Friendly Menu Interface

The program provides an interactive, text-based menu that allows users to perform various operations, such as:

- Converting currency between two countries
- Adding new countries and exchange rates
- Updating exchange rates for existing countries
- Deleting countries from the list
- Inserting new countries at specific positions

The menu-driven approach ensures that users can easily navigate through different functionalities, making the program intuitive for users with minimal technical experience.

Testing Example:

- Adding a new country ("Canada", exchange rate 0.015):
    - The system confirms: "Country added successfully."
- Updating the exchange rate of an existing country ("Japan", new rate 1.50):
    - The system confirms: "Exchange rate updated successfully."
- Deleting an existing country ("UK"):
    - The system confirms: "Country deleted successfully."

Each action performed is confirmed with a message, providing immediate feedback to the user and enhancing user experience.

3. Comparison with Existing Methods

To understand the effectiveness of the implemented solution, it is important to compare it with existing currency converter applications, both online and offline.

3.1 Existing Online Currency Converters

Popular online currency converter platforms like **Google Currency Converter**, **XE Currency**, and **OANDA** offer real-time exchange rate conversion and extensive coverage of global currencies. These platforms use real-time APIs to fetch the latest exchange rates, providing accurate and up-to-date information.

Comparison:

- **Advantages of Online Converters**:
    - **Real-Time Data**: Fetches current exchange rates from financial markets.
    - **Extensive Coverage**: Supports a wide range of currencies, including minor and exotic currencies.
    - **User Experience**: Offers a web-based or mobile interface for ease of access.
- **Disadvantages**:
    - **Internet Dependency**: Requires an internet connection for real-time updates.
    - **Complexity**: May be overwhelming for users who only need basic conversion without detailed market data.

Proposed System vs. Online Converters:

- The proposed system provides a simpler, offline solution that can perform basic currency conversion without requiring internet access. This is beneficial in scenarios where users need a quick conversion tool without relying on external data sources.

- However, it lacks the real-time accuracy of online converters, as the exchange rates are static and predefined by the user.

3.2 Existing Offline Currency Converter Applications

Offline applications, such as mobile apps or desktop programs, store exchange rates locally and allow users to perform conversions without internet access. These apps typically include features for updating rates manually or periodically.

Comparison:

- **Advantages of Offline Apps**:
    - **Offline Functionality**: Can be used without an internet connection.
    - **Data Persistence**: Maintains user-defined exchange rates across sessions.
- **Disadvantages**:
    - **Manual Updates**: Requires users to manually update exchange rates, which may lead to outdated data.

Proposed System vs. Offline Apps:

- The proposed Java-based system aligns closely with offline currency converter applications, offering similar functionality without requiring persistent data storage. The main limitation is that the program does not save data across sessions, meaning users lose their input when the program is closed.
- This could be addressed in future improvements by incorporating a data persistence feature using a file system or database, allowing users to store and retrieve exchange rates across multiple sessions.

4. Discussion of Strengths and Limitations

4.1 Strengths of the Proposed System

- **Ease of Use**: The console-based interface is simple and easy to navigate, making it accessible to users without extensive technical skills.
- **Modular Design**: The program is divided into clear, modular methods, facilitating maintenance and future enhancements.
- **Customizability**: Users can manually add or update countries and exchange rates as needed, allowing for a high degree of flexibility.

4.2 Limitations of the Proposed System

- **Lack of Real-Time Exchange Rates**: The program relies on predefined, static exchange rates. It does not fetch real-time rates, limiting its accuracy in fluctuating currency markets.
- **No Data Persistence**: The system does not save user input across sessions. This means that any changes made to exchange rates or country lists are lost upon exiting the program.
- **Limited User Interface**: The console-based UI, while functional, is less user-friendly than graphical interfaces available in modern applications.

5. Future Improvements

To enhance the proposed currency converter system and bring it closer in functionality to existing solutions, several improvements could be considered:

1. **Integration with Real-Time API**: By connecting to a real-time exchange rate API (e.g., Open Exchange Rates or CurrencyLayer), the program can fetch current rates automatically, improving the accuracy of conversions.
2. **Data Persistence**: Implementing data storage using a file system (CSV, JSON) or a database (SQLite) would allow users to save their custom exchange rates and retrieve them across sessions.

3. **Graphical User Interface (GUI)**: Developing a GUI using JavaFX or Swing would provide a more modern and user-friendly interface, making the program more accessible and visually appealing.

4. **Enhanced Error Handling**: Introducing more robust input validation and error handling would prevent issues related to invalid inputs and improve the overall stability of the program.

5. The results of the proposed currency converter program demonstrate its ability to perform basic currency conversions and manage exchange rates effectively. While it provides a solid offline solution for quick and simple conversions, it falls short when compared to existing online and offline converters that offer real-time updates and persistent data storage. The proposed enhancements would address these limitations, making the application more robust and practical for users seeking a reliable and user-friendly currency conversion tool.
6. By implementing these improvements, the currency converter can evolve from a basic educational program to a fully functional tool suitable for real-world usage in various scenarios, from travel and business to academic purposes.

CONCLUSION

The **Currency Converter** project, developed in Java, serves as a practical and educational tool designed to simplify currency conversion tasks. It offers a fundamental approach to converting currencies using predefined exchange rates, allowing users to calculate the equivalent value of an amount between different currencies. The main aim of this project was to create an offline, user-friendly application that provides basic conversion functionality without relying on internet connectivity.

The program features a menu-driven interface that guides users through various operations, such as converting currencies, adding new countries, updating exchange rates, deleting countries, and inserting new entries at specific positions. By utilizing Java's **ArrayList**, the system effectively manages dynamic data, providing flexibility in handling the list of supported currencies. This design not only makes the program efficient but also offers an easy way for users to update the data as needed.

While the project meets its primary objectives, it has certain limitations. The absence of real-time exchange rate updates is a significant drawback, as it relies on static user-provided rates that may become outdated. Additionally, the lack of data persistence means that changes made by users are not retained between sessions, reducing the convenience of the application for regular users.

To overcome these limitations, future enhancements could include integrating real-time exchange rate APIs to fetch current rates automatically and implementing persistent data storage using files or databases to save user preferences and exchange rates across sessions. A graphical user interface (GUI) could also be introduced to improve user experience and make the application more accessible.

SAMPLE CODING

```
import java.util.ArrayList;
import java.util.Scanner;
public class CurrencyConverter {
    private ArrayList<String> countries;`
    private ArrayList<Double> exchangeRates;
    public CurrencyConverter() {
        countries = new ArrayList<>();
        exchangeRates = new ArrayList<>();
    }
    public double convert(String from, String to, double amount) {
        int fromIndex = countries.indexOf(from);
        int toIndex = countries.indexOf(to);
        if (fromIndex == -1 || toIndex == -1) {
            System.out.println("Invalid country.");
            return -1;
        }
        double fromRate = exchangeRates.get(fromIndex);
        double toRate = exchangeRates.get(toIndex);
        return (amount / fromRate) * toRate;
    }
    public void addCountry(String name, double rate) {
        countries.add(name);
        exchangeRates.add(rate);
        System.out.println("Country added successfully.");
    }
    public void updateRate(String country, double newRate) {
        int index = countries.indexOf(country);
        if (index != -1) {
            exchangeRates.set(index, newRate);
            System.out.println("Exchange rate updated successfully.");
        } else {
            System.out.println("Country not found.");
        }
    }
    public void deleteCountry(String country) {
        int index = countries.indexOf(country);
        if (index != -1) {
            countries.remove(index);
            exchangeRates.remove(index);
            System.out.println("Country deleted successfully.");
        } else {
            System.out.println("Country not found.");
        }
    }
```

```
public void insertCountry(int position, String name, double rate) {
    if (position >= 0 && position <= countries.size()) {
        countries.add(position, name);
        exchangeRates.add(position, rate);
        System.out.println("Country inserted successfully.");
    } else {
        System.out.println("Invalid position.");
    }
}
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    CurrencyConverter converter = new CurrencyConverter();
    converter.addCountry("USA", 0.012);
    converter.addCountry("UK", 0.010);
    converter.addCountry("EURO", 0.011);
    converter.addCountry("Japan", 1.45);
    converter.addCountry("Australia", 0.018);
    while (true) {
        System.out.println("\n1. Convert Currency");
        System.out.println("2. Add Country");
        System.out.println("3. Update Exchange Rate");
        System.out.println("4. Delete Country");
        System.out.println("5. Insert Country");
        System.out.println("6. Exit");
        System.out.print("Choose an option: ");
        int choice = scanner.nextInt();
        switch (choice) {
            case 1:
                System.out.print("Enter from country: ");
                String from = scanner.next();
                System.out.print("Enter to country: ");
                String to = scanner.next();
                System.out.print("Enter amount in rupees: ");
                double amount = scanner.nextDouble();
                double result = converter.convert(from, to, amount);
                if (result != -1) {
                    System.out.printf("Converted amount: %.2f\n", result);
                }
                break;
            case 2:
                System.out.print("Enter country name: ");
                String name = scanner.next();
                System.out.print("Enter exchange rate: ");
                double rate = scanner.nextDouble();
                converter.addCountry(name, rate);
                break;
```

```
            case 3:
                System.out.print("Enter country name to update: ");
                String updateCountry = scanner.next();
                System.out.print("Enter new exchange rate: ");
                double newRate = scanner.nextDouble();
                converter.updateRate(updateCountry, newRate);
                break;
            case 4:
                System.out.print("Enter country name to delete: ");
                String deleteCountry = scanner.next();
                converter.deleteCountry(deleteCountry);
                break;
            case 5:
                System.out.print("Enter position to insert: ");
                int position = scanner.nextInt();
                System.out.print("Enter country name: ");
                String insertCountry = scanner.next();
                System.out.print("Enter exchange rate: ");
                double insertRate = scanner.nextDouble();
                converter.insertCountry(position, insertCountry, insertRate);
                break;
            case 6:
                System.out.println("Exiting...");
                scanner.close();
                return;
            default:
                System.out.println("Invalid choice. Try again.");
        }
    }
  }
}
```

Faculty Incharge **Academic Coordinator** **HOD**