UNIVERSITY INSTITUTE of COMPUTING
Asia's Fastest Growing University

NAAC GRADE A+ ACCREDITED UNIVERSITY

A PROJECT REPORT

PYTHON

MASTER OF COMPUTER APPLICATION

SUBMITTED BY

MD SHAHJAD RAJA

UID – 24MCA20291

CHANDIGARH UNIVERSITY, MASTER OF COMPUTER APPLICATION,

NORTH CAMPUS, PUNJAB(INDIA)

25 OCTOBER 2024

CHANDIGARH UNIVERSITY

Discover. Learn. Empower.

BANK MANAGEMENT SYSTEM USING Tkinter

INDEX PAGE

ABSTRACT

This project focuses on the development of a **Personal Banking System** using Python's **Tkinter** library for creating a graphical user interface (GUI). The system allows users to perform basic banking operations, including creating an account, depositing money, withdrawing funds, and checking the account balance. Each account is uniquely identified by an account number, and all transactions are securely managed within the system.

The primary goal of this project is to simulate a simple banking environment where users can interact with the application in a straightforward, intuitive manner. The project emphasizes the importance of using **object-oriented programming (OOP)** for managing accounts and transactions, ensuring that each account is treated as an independent object with its own properties and behaviours.

Key aspects like **data validation**, **security**, and a **user-friendly interface** were also considered during the development to ensure the system is both functional and reliable. The final application provides a practical demonstration of how Python can be used to create real-world applications that involve managing data and handling user inputs.

This project is beneficial for students and developers looking to understand the basics of banking operations and GUI programming in Python, offering a foundation that can be expanded into more complex banking systems in the future.

INTRODUCTION

In today's digital world, banking systems are very important for managing money and financial transactions. With advancements in technology, people now prefer to handle their banking needs through applications instead of going to physical bank branches. This project aims to create a **simple personal banking system** using **Python** and its **Tkinter** library for building a graphical user interface (GUI).

The personal banking system will allow users to perform basic tasks like creating a bank account, depositing money, withdrawing cash, and checking their account balance. Each account will have a unique account number to keep track of individual accounts securely.

The main idea behind this personal banking system is to provide users with a straightforward way to manage their banking tasks without needing to visit a bank. Users will be able to create their own bank accounts, deposit money into them, withdraw cash when needed, and check how much money they have in their accounts. Each account will have a unique account number, ensuring that all transactions are secure and easily identifiable. This approach not only makes banking more accessible but also empowers individuals to take control of their finances from the comfort of their own homes.

Using Python for this project is particularly beneficial because it is a versatile programming language that is easy to learn and use. Python's Tkinter library allows us to create a user-friendly interface, making it simple for users to interact with the banking

system. The GUI will include various buttons and input fields that guide users through different banking operations, ensuring that even those with little technical knowledge can navigate the application smoothly.

In addition to creating a user-friendly experience, this project will also emphasize the importance of security and data validation in banking applications. It is crucial to ensure that users cannot perform invalid operations, such as withdrawing more money than they have in their accounts or entering incorrect account details. To address this, the application will implement checks and balances that prompt users to enter valid amounts and provide error messages when necessary.

In conclusion, this introduction outlines the primary goals and objectives of the personal banking system project. By leveraging Python and Tkinter, we aim to create a simple, secure, and user-friendly banking application that allows individuals to manage their finances effectively. Through this project, we hope to bridge the gap between technology and everyday financial management, making banking more accessible to everyone.

LITERETURE REVIEW

The development of personal banking systems using programming languages like Python has gained significant attention in recent years. This literature review explores the key concepts and findings from various studies related to digital banking, programming

languages, user interface design, security measures, and object-oriented programming (OOP).

1. Digital Banking Importance

Digital banking has become essential as more people prefer managing their finances online rather than visiting physical banks. Research highlights how digital banking enhances customer satisfaction and improves the efficiency of banking services. As technology advances, there is a growing need for simple and user-friendly banking applications that allow users to perform essential functions like creating accounts, depositing money, and checking balances from anywhere.

2. Programming Languages

Python has emerged as a popular choice for developing banking applications due to its ease of use and versatility. The Tkinter library in Python allows developers to create graphical user interfaces (GUIs) quickly. Studies have shown that while Python is excellent for small to medium-sized applications, other languages like Java might be better suited for larger, more complex banking systems. This indicates that the choice of programming language can significantly impact the development process.

3. User Interface Design

A well-designed user interface (UI) is crucial for banking applications. Research emphasizes the importance of usability and accessibility, ensuring that users can navigate

the application easily. Studies suggest that incorporating user feedback during the design process can help developers create more effective and intuitive interfaces. A good UI can enhance user engagement and make financial management less stressful for customers.

4. Security Measures

Security is a top priority in banking applications. The literature discusses various security measures, such as encryption and secure authentication, that protect users' sensitive information. With increasing cyber threats, researchers advocate for implementing advanced security technologies, like biometric authentication, to improve account security. Strong security protocols are essential to build trust and ensure users feel safe while using banking applications.

5. Object-Oriented Programming

Using object-oriented programming (OOP) principles in banking applications helps organize the system better. OOP allows developers to represent real-world entities, like accounts and transactions, as objects. This makes the application more manageable and easier to update. By following OOP practices, developers can create a scalable and maintainable banking system, which is vital as user needs evolve.

OBJECTIVES

The primary objectives of this project are:

1. **To develop a personal banking system using Python and Tkinter**: The core objective is to create a simple, functional banking system that allows users to manage their accounts through a graphical interface.
2. **To implement basic banking functionalities**: Users should be able to create an account, deposit funds, withdraw money, and check their account balance in an intuitive and secure environment.
3. **To understand and apply object-oriented programming**: The project utilizes OOP concepts like classes and methods to represent the different elements of the banking system, such as accounts and transactions.
4. **To ensure data validation and security**: Implementing data validation to ensure users do not input incorrect information, and to prevent invalid transactions such as overdrafts.
5. **To build an intuitive user interface**: The Tkinter library is used to create a user-friendly GUI that provides a seamless experience when navigating through the different banking operations.

6. **To enhance programming skills in GUI development**: This project serves as a practical demonstration of how to build a complete application with a focus on user interaction and design principles.

METHODLOGY

The project is developed using the **Tkinter** library for creating the graphical user interface and **Python** for handling the backend logic of the banking system. The following steps were followed in the development of this project:

1. **Planning and Design**:
    - Identifying the features and operations of the banking system.
    - Designing the user interface layout for ease of use.
    - Deciding on the structure and flow of the program, using OOP concepts.
2. **Development**:
    - Implementing the Account class to handle individual account details and operations.
    - Creating the Bank class to manage multiple accounts and perform actions like account creation and retrieval.
    - Building the GUI using Tkinter widgets such as Label, Button, Entry, and Toplevel to facilitate user interaction.

3. **Testing and Validation**:
    - Testing each operation (deposit, withdrawal, account creation) with valid and invalid inputs to ensure error handling works.
    - Checking if the GUI responds as expected when interacting with different functions.
4. **Deployment**:
    - Integrating all features into a final application that can be executed as a standalone desktop program.

DESCRIPTION

1. Account Class:

- This class represents a bank account. It has attributes like account_number, account_holder, and balance. The class includes methods for depositing money, withdrawing money, and displaying the current balance.

2. Bank Class:

- The Bank class manages multiple accounts. It stores accounts in a dictionary, with each account's number serving as the key. It includes methods for creating a new account and retrieving an existing account.

3. BankApp Class:

- This class builds the GUI using Tkinter. It provides an interface for users to create accounts, deposit funds, withdraw money, and check account balances. The class uses several Tkinter widgets to enable user interaction.

4. User Interface:

- The Tkinter library is used to create a simple and interactive interface. Each banking operation has a dedicated window where users can input the necessary details (such as account number and amount) and receive feedback on the success of the operation.

CODE IMPLEMENTION

```
import tkinter as tk
from tkinter import messagebox


class Account:
    def __init__(self, account_number, account_holder, balance=0):
        self.account_number = account_number
        self.account_holder = account_holder
        self.balance = balance

    def deposit(self, amount):
        if amount <= 0:
            return "Please enter a valid deposit amount."
        self.balance += amount
        return f"Deposited: {amount}. New Balance: {self.balance}"

    def withdraw(self, amount):
        if amount <= 0:
            return "Please enter a valid withdrawal amount."
        if amount > self.balance:
            return "Insufficient balance!"
        self.balance -= amount
        return f"Withdrew: {amount}. New Balance: {self.balance}"

    def display_balance(self):
        return f"Account Number: {self.account_number}, Balance: {self.balance}"


class Bank:
    def __init__(self):
        self.accounts = {}

    def create_account(self, account_number, account_holder):
        if account_number in self.accounts:
            return "Account already exists!"
        else:
            new_account = Account(account_number, account_holder)
```

```
            self.accounts[account_number] = new_account
            return "Account created successfully!"

    def get_account(self, account_number):
        return self.accounts.get(account_number, None)


class BankApp:
    def __init__(self, root):
        self.bank = Bank()
        self.root = root
        self.root.title("Bank Management System")
        self.root.geometry("500x400")

        self.title_label = tk.Label(root, text="Bank Management System", font=("Arial",
16, "bold"))
        self.title_label.pack(pady=10)

        self.create_account_btn = tk.Button(root, text="Create Account", width=20,
command=self.create_account_ui)
        self.create_account_btn.pack(pady=5)

        self.deposit_btn = tk.Button(root, text="Deposit", width=20,
command=self.deposit_ui)
        self.deposit_btn.pack(pady=5)

        self.withdraw_btn = tk.Button(root, text="Withdraw", width=20,
command=self.withdraw_ui)
        self.withdraw_btn.pack(pady=5)

        self.check_balance_btn = tk.Button(root, text="Check Balance", width=20,
command=self.check_balance_ui)
        self.check_balance_btn.pack(pady=5)

        self.exit_btn = tk.Button(root, text="Exit", width=20, command=root.quit)
        self.exit_btn.pack(pady=5)

    def create_account_ui(self):
        account_window = tk.Toplevel(self.root)
        account_window.title("Create Account")
```

```
        account_window.geometry("400x250")

        tk.Label(account_window, text="Account Number:").pack(pady=5)
        account_number_entry = tk.Entry(account_window)
        account_number_entry.pack(pady=5)

        tk.Label(account_window, text="Account Holder:").pack(pady=5)
        account_holder_entry = tk.Entry(account_window)
        account_holder_entry.pack(pady=5)

        def create():
            account_number = account_number_entry.get()
            account_holder = account_holder_entry.get()
            message = self.bank.create_account(account_number, account_holder)
            messagebox.showinfo("Create Account", message)
            account_window.destroy()

        tk.Button(account_window, text="Create", command=create).pack(pady=20)

    def deposit_ui(self):
        deposit_window = tk.Toplevel(self.root)
        deposit_window.title("Deposit")
        deposit_window.geometry("400x250")

        tk.Label(deposit_window, text="Account Number:").pack(pady=5)
        account_number_entry = tk.Entry(deposit_window)
        account_number_entry.pack(pady=5)

        tk.Label(deposit_window, text="Amount:").pack(pady=5)
        amount_entry = tk.Entry(deposit_window)
        amount_entry.pack(pady=5)

        def deposit():
            account_number = account_number_entry.get()
            amount = float(amount_entry.get())
            account = self.bank.get_account(account_number)
            if account:
                message = account.deposit(amount)
                messagebox.showinfo("Deposit", message)
            else:
```

```
            messagebox.showerror("Error", "Account not found!")
        deposit_window.destroy()

    tk.Button(deposit_window, text="Deposit", command=deposit).pack(pady=20)

def withdraw_ui(self):
    withdraw_window = tk.Toplevel(self.root)
    withdraw_window.title("Withdraw")
    withdraw_window.geometry("400x250")

    tk.Label(withdraw_window, text="Account Number:").pack(pady=5)
    account_number_entry = tk.Entry(withdraw_window)
    account_number_entry.pack(pady=5)

    tk.Label(withdraw_window, text="Amount:").pack(pady=5)
    amount_entry = tk.Entry(withdraw_window)
    amount_entry.pack(pady=5)

    def withdraw():
        account_number = account_number_entry.get()
        amount = float(amount_entry.get())
        account = self.bank.get_account(account_number)
        if account:
            message = account.withdraw(amount)
            messagebox.showinfo("Withdraw", message)
        else:
            messagebox.showerror("Error", "Account not found!")
        withdraw_window.destroy()

    tk.Button(withdraw_window, text="Withdraw", command=withdraw).pack(pady=20)

def check_balance_ui(self):
    balance_window = tk.Toplevel(self.root)
    balance_window.title("Check Balance")
    balance_window.geometry("400x200")

    tk.Label(balance_window, text="Account Number:").pack(pady=5)
    account_number_entry = tk.Entry(balance_window)
    account_number_entry.pack(pady=5)
```

```
        def check_balance():
            account_number = account_number_entry.get()
            account = self.bank.get_account(account_number)
            if account:
                message = account.display_balance()
                messagebox.showinfo("Balance", message)
            else:
                messagebox.showerror("Error", "Account not found!")
            balance_window.destroy()

        tk.Button(balance_window, text="Check Balance",
command=check_balance).pack(pady=20)


if __name__ == "__main__":
    root = tk.Tk()
    app = BankApp(root)
    root.mainloop()
```

OUTPUT


Bank Management System
Bank Management System
Create Account
Deposit
Withdraw
Check Balance
Exit

RESULT

The final result of the project is a fully functional **bank management system** with the following outcomes:

1. **Account Creation**: Users can successfully create an account by providing an account number and the name of the account holder. Duplicate account numbers are detected and prevented.
2. **Deposits**: The system allows users to deposit money into their account, updating the balance accurately. Input validation ensures only positive amounts can be deposited.
3. **Withdrawals**: The withdrawal functionality works as expected, allowing users to withdraw funds while checking for sufficient balance. Invalid inputs and overdrafts are prevented.
4. **Balance Check**: Users can view their account balance at any time, receiving clear and accurate information about their available funds.
5. **User Interface**: The Tkinter-based GUI is intuitive, allowing users to navigate easily between different banking operations.

CONCLUSION

The project demonstrates how to create a simple yet functional banking system using Python and the Tkinter library. By utilizing object-oriented programming principles, we were able to create a secure and user-friendly system that allows users to manage basic banking operations efficiently. The project emphasizes the importance of error handling, data validation, and a smooth user interface for successful software development. While the system is basic in terms of features, it provides a solid foundation for more complex banking applications and demonstrates key principles of GUI design and programming.

Overall, this project serves as a practical demonstration of how technology can simplify banking processes and enhance financial management for individuals. By leveraging the insights gained from this project, developers can build upon this foundation to create more advanced banking systems that address the evolving demands of users. The personal banking system not only meets current needs but also lays the groundwork for future developments in digital banking, making financial management more accessible and efficient for everyone.

REFERENCES

1. **Python Documentation**: https://docs.python.org/3/
   (For learning about the Python language and Tkinter library.)
2. **Tkinter Reference**: https://tkdocs.com/
   (For detailed tutorials on building GUIs using Tkinter.)
3. **Object-Oriented Programming in Python**: https://realpython.com/python3-object-oriented-programming/
   (For understanding OOP concepts in Python.)
4. **Banking System Example Projects**:
   (For reference code and ideas for implementing similar projects.)