**Project Description for the Bank System with OOP and Exception Handling**

**1. Aim of the Project**

The primary aim of this project is to develop a simple, efficient banking system using Object-Oriented Programming (OOP) concepts in Python. The system is designed to handle basic banking operations, including deposit, withdrawal, and balance checking, while ensuring user input validation and robust error handling. By implementing custom exceptions, the project seeks to provide a smooth user experience by gracefully handling invalid inputs and preventing system crashes. This project demonstrates the practical application of OOP principles and showcases the importance of error handling in real-world software systems.

**2. Business Problem or Problem Statement**

In many small businesses and personal financial systems, managing multiple accounts with proper validation and error handling can be a challenge. Manual systems are prone to errors, such as incorrect balance calculations or transactions performed without sufficient funds. Additionally, handling invalid user inputs (e.g., negative amounts or invalid account numbers) can cause system crashes or inconsistencies, which could disrupt the financial processes and user experience.

This project addresses these issues by providing a digital banking solution that automates account management and ensures accurate, validated transactions. By introducing a user-friendly interface with built-in error handling mechanisms, this system minimizes the risk of human error, prevents financial miscalculations, and ensures that transactions are conducted securely. The primary problem the project tackles is creating a system where common banking operations can be performed seamlessly, with robust error handling to manage invalid inputs or insufficient funds, without breaking the system.

**3. Project Description**

The Banking System project is a Python-based application designed to demonstrate the use of Object-Oriented Programming (OOP) principles and exception handling in real-world scenarios. The project revolves around managing multiple bank accounts and performing basic operations such as deposits, withdrawals, and balance checks. The system is built to be interactive and user-friendly, ensuring that all user inputs are properly validated before processing any transaction.

**Scope**: The scope of the project includes creating a multi-account system where users can perform banking transactions efficiently. The system is designed to manage individual accounts and handle the flow of transactions while preventing invalid actions through error handling. This project provides a foundational framework for banking systems, focusing on essential banking functions while leaving room for future enhancements such as adding account transfers or more advanced security measures.

**Objectives**: The key objective of this project is to create a banking system that allows users to interact with their accounts using deposit, withdrawal, and balance check features. The secondary objective is to ensure that all invalid inputs are handled smoothly by using custom exceptions and built-in error handling to prevent crashes or incorrect transactions.

**Technologies Used**:

* **Python**: The primary programming language used for the implementation.
* **OOP Concepts**: The project heavily relies on OOP, with the use of classes (BankAccount and Bank) to model the real-world banking system.
* **Exception Handling**: Custom exceptions like InsufficientBalanceError and InvalidAccountError are used to handle errors such as insufficient funds or invalid account entries.

**Key Features**:

* Multi-account management.
* Transaction handling (deposit, withdrawal, balance check).
* Input validation and error handling.

The system is structured around two main classes: the BankAccount class that represents individual accounts, and the Bank class that manages multiple accounts. The project also includes detailed error handling for common scenarios, such as attempting to withdraw more than the available balance or entering invalid inputs for transaction amounts.

**4. Functionalities**

1. **Deposit Functionality**:
   * **Description**: This functionality allows users to deposit a specified amount into their bank account. It checks if the deposit amount is valid (i.e., greater than zero). If valid, the amount is added to the account balance, and a confirmation message is displayed.
2. **Withdrawal Functionality**:
   * **Description**: This feature enables users to withdraw money from their account, provided they have sufficient funds. The system checks if the withdrawal amount is valid and if the account balance is sufficient before proceeding. If either condition fails, a custom exception is triggered, notifying the user of insufficient balance.
3. **Balance Check**:
   * **Description**: This functionality allows users to view their current account balance at any time. It displays the balance in a user-friendly format without making any changes to the account.
4. **Input Validation and Error Handling**:
   * **Description**: The system performs validation on all user inputs, including checking for non-numeric entries, negative amounts, and invalid account numbers. This ensures that only valid operations are processed, and it provides informative error messages when invalid inputs are detected.
5. **Exit Functionality**:
   * **Description**: This option allows users to safely exit the system after completing their transactions. The program closes after printing a thank-you message.

**5. Input Versatility with Error Handling and Exception Handling**

The system is designed to handle a wide variety of user inputs with built-in versatility and error handling mechanisms. For example, when the user attempts to deposit or withdraw money, the system ensures that the input is a positive number. If the user tries to enter non-numeric values or negative numbers, a ValueError is raised, and the system prompts the user to enter valid data.

**Custom Exception Handling**:

* The system includes custom exceptions like InsufficientBalanceError to ensure that users cannot withdraw more than their available balance. If such an attempt is made, the system displays an error message and prevents the transaction from going through.
* Similarly, InvalidAccountError handles cases where the user enters an invalid account number. This provides a smooth and error-free experience without causing system crashes.

**6. Code Implementation**

The project is implemented using Python’s Object-Oriented Programming (OOP) model, which helps in organizing code efficiently. The project consists of two main classes: BankAccount and Bank.

**Code Structure**:

* **BankAccount Class**: Manages individual account-related operations like deposits, withdrawals, and balance checks. It holds key attributes such as account\_number, owner, and balance.
* **Bank Class**: Acts as a manager for multiple BankAccount objects. It contains methods for adding new accounts and retrieving accounts based on account numbers.

**Key Algorithms**:

* Input validation ensures that all inputs are valid before any transaction is processed.
* The system implements algorithms to check for sufficient balance before withdrawing and ensures that deposit amounts are valid.

**7. Results and Outcomes**

The project successfully implements a banking system that can handle multiple accounts and perform transactions smoothly. Key outcomes include:

* Robust error handling ensures that invalid inputs do not cause crashes.
* The system is capable of handling basic banking functionalities like deposits, withdrawals, and balance checks with high reliability.

**8. Conclusion**

This project demonstrates the practical application of OOP concepts and exception handling in creating a reliable banking system. The project’s significance lies in its ability to provide a smooth user experience while ensuring that errors are handled effectively. Future enhancements could include adding features such as account transfers or enhanced security protocols for financial transactions.