**Exercise 5- A simple banking app using exceptions**

**Introduction**

This report provides an overview of the Java-based Bank Account Management System, detailing the code structure, functionality, and the underlying thought process. The program simulates basic banking operations, allowing users to create an account, deposit funds, and withdraw funds while ensuring data integrity through proper validation and exception handling.

**Code Structure**

The program consists of three primary components:

1. **Custom Exception Class (InsufficientFundsException)**
2. **Main Business Logic Class (Bank\_Account)**
3. **Application Entry Point (App)**

**1. InsufficientFundsException Class**

This class extends Exception and is used to handle cases where a user attempts to withdraw an amount exceeding their current balance. The class constructor takes a message parameter, allowing meaningful error messages to be displayed.

**2. Bank\_Account Class**

This class represents a bank account with essential attributes:

* accountNumber: A unique identifier for the account.
* accountHolder: The name of the account owner.
* balance: The amount of money currently in the account.

**Methods:**

* **Constructor**: Initializes the account with provided values.
* **Getters**: Retrieve account information.
* **deposit(double amount)**: Increases the account balance.
* **withdrawing(double amount)**: Deducts the specified amount from the balance without additional validation.

**3. App Class (Main Application Logic)**

This class serves as the entry point and manages user interaction via the console.

**Key Features:**

* **Account Creation (createAccount)**:
  + Prompts the user to enter account details.
  + Ensures the initial balance is greater than zero.
* **Banking Operations (Depositing & Withdrawing)**:
  + Uses a loop to present a menu allowing the user to choose between deposit, withdrawal, or exiting.
  + **Depositing (amountValidation)**:
    - Ensures the deposit amount is greater than zero.
    - Throws an IllegalArgumentException if invalid input is detected.
  + **Withdrawing (amountWithdrawingValidation)**:
    - Ensures the withdrawal amount is valid.
    - Throws an IllegalArgumentException for non-positive amounts.
    - Throws InsufficientFundsException if the requested amount exceeds the balance.
* **Exception Handling**:
  + Prevents invalid transactions using try-catch blocks in the main loop.
  + Displays meaningful error messages to the user.

**Thought Process**

The approach to designing this program is structured as follows:

1. **Encapsulation & Modularity**:
   * The Bank\_Account class encapsulates data, keeping it secure and accessible only via public methods.
   * Separate methods for validation and account creation enhance modularity and maintainability.
2. **User Input Validation & Exception Handling**:
   * Ensuring valid inputs reduces runtime errors.
   * Using IllegalArgumentException and InsufficientFundsException helps maintain system stability and inform users of issues.
3. **Flexibility & Scalability**:
   * The menu-driven approach allows easy expansion (e.g., adding more banking operations).
   * The exception-handling mechanism ensures robust error handling for future improvements.

**Conclusion**

The program effectively simulates a simple banking system, focusing on structured design, error handling, and a user-friendly interface. It ensures valid transactions and prevents common input-related errors, demonstrating best practices in object-oriented programming and exception handling.

Further improvements could include:

* Adding file-based data persistence.
* Enhancing security measures (e.g., PIN verification).
* Implementing additional banking features such as fund transfers.

Overall, the program is well-structured, demonstrating clear logical thinking and adherence to programming best practices.