DETAILED EXPLANATION OF THE ATM PROGRAM (OOP in Python)

1. OBJECT-ORIENTED DESIGN:
   - We define a class `ATM` that encapsulates the state (balance) and behaviors (check_balance, deposit, withdraw) of an ATM account.
   - This follows the **Encapsulation** principle in OOP, where data and related methods are bundled together.

2. CONSTRUCTOR (`__init__`):
   - The constructor initializes the ATM object with a starting balance.
   - Default balance is set to 0 if not provided.
   - Time complexity: O(1) — constant time initialization.

3. METHODS:
   a) `check_balance()`:
      - Prints the current balance formatted to 2 decimal places.
      - Time complexity: O(1).

   b) `deposit(amount)`:
      - Validates that the deposit amount is positive.
      - If valid, adds the amount to the balance.
      - Uses floating-point arithmetic; amounts are displayed in standard currency format.
      - Time complexity: O(1).

   c) `withdraw(amount)`:
      - Validates that the withdrawal amount is positive and does not exceed the current balance.
      - Deducts the amount if conditions are met.
      - Time complexity: O(1).

4. MAIN PROGRAM LOOP:
   - Implements a **menu-driven interface** using a `while True` loop.
   - Displays options: Check Balance, Deposit, Withdraw, Exit.
   - Uses `try-except` blocks to handle invalid numeric inputs.
   - Breaks the loop when the user chooses to exit.

```
5. FORMAL ALGORITHMIC APPROACH:
   Let:
     - B = current balance
     - A = transaction amount
   Algorithm:
     1. Initialize B with starting balance.
     2. Repeat until user chooses Exit:
        a. Display menu.
        b. If choice = 1 → Output B.
        c. If choice = 2 → If A > 0, set B = B + A.
        d. If choice = 3 → If 0 < A ≤ B, set B = B - A.
        e. Else → Display error.
     3. End.

6. SCIENTIFIC NOTATION:
   - Balances and amounts are stored as floating-point numbers.
   - Python internally uses IEEE 754 double-precision floating-point representation.
   - Example: $1,000 is stored as 1.0 × 10³ in scientific notation internally.

7. EDGE CASES HANDLED:
   - Negative or zero deposit/withdrawal amounts.
   - Withdrawal exceeding balance.
   - Non-numeric input for amounts.
   - Invalid menu choices.
```

```python
class ATM:
    def __init__(self, balance=0):
        """Constructor to initialize ATM with a starting balance."""
        self.balance = balance

    def check_balance(self):
        """Display the current balance."""
        print(f"Your current balance is: ${self.balance:.2f}")
```

```
    def deposit(self, amount):
        """Deposit money into the account."""
        if amount <= 0:
            print("Deposit amount must be positive.")
            return
        self.balance += amount
        print(f"${amount:.2f} deposited successfully.")

    def withdraw(self, amount):
        """Withdraw money from the account."""
        if amount <= 0:
            print("Withdrawal amount must be positive.")
        elif amount > self.balance:
            print("Insufficient funds.")
        else:
            self.balance -= amount
            print(f"${amount:.2f} withdrawn successfully.")

# -------- Main Program --------
def main():
    atm = ATM(balance=1000)  # Start with $1000
    while True:
        print("\n===== ATM Menu =====")
        print("1. Check Balance")
        print("2. Deposit")
        print("3. Withdraw")
        print("4. Exit")

        choice = input("Enter your choice: ")

        if choice == "1":
            atm.check_balance()
        elif choice == "2":
            try:
                amount = float(input("Enter deposit amount: "))
                atm.deposit(amount)
            except ValueError:
```

```
                print("Invalid amount. Please enter a number.")
        elif choice == "3":
            try:
                amount = float(input("Enter withdrawal amount: "))
                atm.withdraw(amount)
            except ValueError:
                print("Invalid amount. Please enter a number.")
        elif choice == "4":
            print("Thank you for using the ATM. Goodbye!")
            break
        else:
            print("Invalid choice. Please try again.")

if __name__ == "__main__":
    main()
```

If you want, I can now **extend this ATM** to include **PIN authentication and transaction history** so it behaves more like a real-world ATM system. Would you like me to do that?