กิจกรรมที่ 4 : Class Diagram

1. ให้นักศึกษาออกแบบและพัฒนาโปรแกรมระบบตู้ ATM ของธนาคารแห่งหนึ่ง โดยมีข้อกำหนดดังนี้
    - **ธนาคาร (Bank):** ให้บริการ User เปิดบัญชี โดย User 1 คนจะเปิดกี่บัญชีก็ได้
        - ข้อมูล User: เลขบัตรประชาชน, ชื่อ–นามสกุล
        - ข้อมูล Account: หมายเลขบัญชี, เจ้าของบัญชี (User instance), ยอดเงินคงเหลือ
    - **บัตร ATM:** ใน 1 บัญชี สามารถผูกบัตร ATM ได้ 1 ใบ
        - ข้อมูลบัตร: หมายเลขบัตร, หมายเลขบัญชีที่ผูก, รหัสผ่าน (PIN)
    - **ตู้ ATM (ATM Machine)**:
        - ข้อมูลตู้: หมายเลขตู้ (ATM ID), จำนวนเงินสดที่บรรจุในตู้ (ไม่เท่ากันในแต่ละตู้)
        - ตู้ ATM ทำหน้าที่เป็น Interface ระหว่างผู้ใช้กับระบบธนาคาร
    - **กฎเกณฑ์และค่าธรรมเนียม**:
        - ค่าธรรมเนียมรายปี: 150 บาท (เก็บเป็นค่าคงที่ในระบบ)
        - วงเงินถอนสูงสุดต่อวัน (Daily Limit): ห้ามถอนเกิน 40,000 บาท/บัญชี/วัน (รวมทุกธุรกรรมการถอนและการโอนออก)
    - **ธุรกรรม (Transactions):** การทำรายการผ่านตู้ ATM ต้องรองรับ:
        - การฝากเงิน (Deposit)
        - การถอนเงิน (Withdraw): ต้องตรวจสอบวงเงินต่อวัน และ ตรวจสอบว่าตู้ ATM มีเงินสดเพียงพอหรือไม่
        - การโอนเงิน (Transfer): ต้องตรวจสอบวงเงินต่อวัน
    - **การบันทึกรายการ (Transaction Logging):** ทุกครั้งที่ทำธุรกรรมสำเร็จ ต้องบันทึกประวัติลงใน Account นั้นๆ โดยเก็บข้อมูล:
        - ประเภท (D=Deposit, W=Withdraw, TW=Transfer Withdraw [ผู้โอน], TD=Transfer Deposit [ผู้รับโอน])
        - หมายเลขตู้ ATM, จำนวนเงิน, ยอดเงินคงเหลือ, และเลขบัญชีคู่กรณี (ถ้ามี)
2. ข้อกำหนดในการเขียนโปรแกรม
    - **Design First:** ให้เขียน Class Diagram ที่แสดงถึงระบบข้างต้น โดยยังไม่ต้องระบุ Multiplicity และให้เขียน Class Diagram ให้ Staff ตรวจก่อนการเขียนโปรแกรม
    - **ห้ามใช้ Dictionary** ในการเก็บข้อมูลภายใน Class (ให้ใช้ Attribute ของ Object)
    - **Encapsulation:** ข้อมูลภายใน Class (Attributes) ต้องเป็น **Private** ทั้งหมด และสร้าง Getter/Setter เท่าที่จำเป็นต้องใช้งานจริง
    - Validation & Robustness:

- ห้ามมีการรับ Input (input()) หรือแสดงผล (print()) ภายใน Class ที่เป็น Logic (Bank, User, Account, etc.) ให้ทำหน้าที่เป็น Service เท่านั้น
- **Exception Handling:** หากเกิดข้อผิดพลาด (เช่น เงินไม่พอ, PIN ผิด, ค่าติดลบ) **ต้องทำการ Raise Exception** (เช่น ValueError, Exception) ออกมา ห้าม return เป็น String ว่า "Error" เฉยๆ
- **ข้อมูลในระบบต้องมีที่เก็บที่เดียวเท่านั้น** ห้ามมิให้เก็บข้อมูลที่ไม่ใช่ข้อมูลที่เกี่ยวข้องกับคลาสนั้น หากไม่สามารถเก็บลงในคลาสใดได้เลย ให้พิจารณาสร้างคลาสใหม่

3. **Skeleton Code:** ให้ดาวน์โหลดไฟล์ lab4_skeleton.py
   - ให้นักศึกษาเขียน Class Definition ให้ครบถ้วน
   - **ห้ามแก้ไข** Code ในส่วน create_bank_system() และ run_test() โดยเด็ดขาด
   - โปรแกรมที่เขียนต้องทำให้ Test Case ทั้ง 10 ข้อใน run_test() ทำงานได้ถูกต้อง

4. ข้อมูลจำลองสำหรับการทดสอบ (Test Data)
   - **User 1:** Harry Potter (Citizen ID: 1-1101-12345-12-0)

     Account: 1000000001, Balance: 20,000, ATM Card: 12345, PIN: 1234
   - **User 2**: Hermione Granger (Citizen ID: 1-1101-12345-13-0)

     Account: 1000000002, Balance: 1,000, ATM Card: 12346, PIN: 1234
   - **ATM Machine 1**: ID 1001, เงินสดในตู้: 1,000,000 บาท
   - **ATM Machine 2**: ID 1002, เงินสดในตู้: 200,000 บาท

5. **รายการทดสอบ (Test Cases)**

   โปรแกรมต้องผ่านการทดสอบทั้ง 10 กรณี ดังนี้:
   - Test Case #1: ทดสอบการสอดบัตรและตรวจสอบ PIN (ถูกต้อง)
   - Test Case #2: ทดสอบการฝากเงิน (Deposit)
   - Test Case #3: ทดสอบฝากเงินด้วยค่าติดลบ (ต้อง Raise Exception)
   - Test Case #4: ทดสอบการถอนเงิน (Withdraw)
   - Test Case #5: ทดสอบถอนเงินเกินจำนวนที่มีในบัญชี (Overdraft)
   - Test Case #6: ทดสอบการโอนเงิน (Transfer)
   - Test Case #7: แสดงรายการเดินบัญชี (Transaction History) รูปแบบ: [Type]-ATM:[ATM_ID]-[Amount]-[Balance]-[Target]
   - Test Case #8: ทดสอบใส่ PIN ผิด (ต้องปฏิเสธการใช้งาน)
   - Test Case #9: ทดสอบถอนเงินเกินวงเงินจำกัดต่อวัน (40,000 บาท)
   - Test Case #10: ทดสอบถอนเงินเมื่อ เงินสดในตู้ ATM ไม่พอจ่าย

การตรวจ

1. เมื่อเขียน Class Diagram เสร็จ ให้ Staff ตรวจ โดยให้ตอบคำถาม เรื่อง ความสัมพันธ์ระหว่าง Class จาก Class Diagram ที่นักศึกษาออกแบบ ว่าระบุว่าเป็นความสัมพันธ์รูปแบบใด (**Composition** หรือ **Aggregation**) และให้เหตุผลประกอบ
    - Bank กับ ATM_Machine
    - Bank กับ User
    - User กับ Account
    - Account กับ ATM_Card
2. นำโปรแกรมมาเขียนให้ทำงานครบตาม Test Case แล้ว ให้ Staff ตรวจอีกครั้ง และ แจ้งด้วยว่ามีการแก้ไข Class Diagram หรือไม่

คำแนะนำ:

- ควรเริ่มจากการวาด class diagram ให้ครบถ้วนก่อนเริ่มเขียนโค้ด
- ให้ความสำคัญกับการ validate ข้อมูลในทุก method
- ระวังเรื่อง data encapsulation โดยใช้ private attributes
- ควรเพิ่ม type hints เพื่อความชัดเจนของโค้ด