กิจกรรมที่ 5 : Inheritance

1. ให้นำโปรแกรมจากกิจกรรมที่ 4 มาเขียนเพิ่มเติม โดยนำ code เดิมมาขยายเพิ่มเติม
2. **บัญชี** เพิ่มการรองรับบัญชีเป็น 3 ประเภท คือ
    - ออมทรัพย์ (SavingAccount) อัตราดอกเบี้ย 0.5 % ต่อปี ผู้ถือบัญชีสามารถฝากเท่าไรก็ได้ แต่ถอนได้ไม่เกิน 40,000 บาทต่อครั้ง และมีวงเงินถอนรายวันได้ไม่เกิน 40,000 บาท (ยกเว้นมีบัตร Premium Card)
    - ฝากประจำ (FixedAccount) อัตราดอกเบี้ย 2.5 % ต่อปี ต้องฝากเงินตามระยะเวลาที่กำหนด (เช่น 6 เดือน หรือ 1 ปี) หากถอนเมื่อครบกำหนดจะได้รับดอกเบี้ยเต็มจำนวน โดยการคำนวณวันครบกำหนดจะใช้ วันเริ่มต้น + (จำนวนเดือน × 30 วัน) แต่หากถอนก่อนกำหนดจะได้รับดอกเบี้ย 50% (โดยมีการแสดง warning หากถอนก่อนกำหนด) บัญชีประเภทนี้ไม่มีการกำหนดวงเงินการถอน
    - กระแสรายวัน (CurrentAccount) ถอนได้ไม่จำกัดจำนวน, ไม่มีดอกเบี้ย วงเงินถอนรายวัน ไม่เกิน 40,000 บาท (กรณีผ่าน ATM หรือ EDC)
3. **บัตร ATM:** ใน 1 บัญชี สามารถผูกบัตร ATM ได้ 1 ใบ รองรับบัตรดังนี้
    - บัตร ATM ธรรมดา (ATM Card)**
        - ใช้ได้เฉพาะ ตู้ ATM
        - ค่าธรรมเนียมรายปี 100 บาท
    - บัตรเดบิต (Debit Card)
        - ใช้ได้ทั้ง ATM และ EDC (เครื่องรูดบัตร) ค่าธรรมเนียมรายปี: 300 บาท
        - บัตรเดบิต จะมีประเภทย่อยอีก 2 ประเภท โดยนอกจากมีคุณสมบัติข้างต้นแล้ว ยังมีคุณสมบัติเพิ่มเติมอีก
            - บัตรช้อปปิ้ง (Shopping Card) ค่าธรรมเนียมรายปี: 300 บาท จะได้เครดิตเงินคืน 1% (เมื่อจ่ายผ่าน EDC ≥ 1,000 บาท) โดยเครดิตคืนจะเข้าบัญชีทันทีหลังชำระเงิน
            - บัตร Premium ค่าธรรมเนียม 500/ปี, Cashback 2%, วงเงิน 100,000/วัน
4. **ช่องทางทำธุรกรรม** เพิ่มเป็น 3 ช่องทาง
    - **ตู้ ATM (ATM Machine)**:
        - รองรับบัตรทุกประเภท
        - เมื่อใช้งานต้องมีการ ใส่บัตร และ ตรวจสอบ PIN จึงสามารถทำรายการได้
        - รองรับการ **นำบัตรออก** และระหว่างทำรายการต้องมีบัตรในเครื่อง
        - การถอนเงินในช่องทางนี้จะไม่สามารถถอนจนหมดบัญชีได้ ต้องเหลือเงินมากกว่าค่าธรรมเนียมรายปีเสมอ

    - การถอนผ่านตู้ จะถอนได้ครั้งละไม่เกิน 40,000 บาท และต้องตรวจสอบเงินในตู้ว่ามีเพียงพอ
- **เคาน์เตอร์ (Counter)**
    - ไม่ต้องใช้บัตร
    - ยืนยันตัวตนด้วย บัตรประจำตัวประชาชน โดยระหว่างการทำธุรกรรมจะต้องเก็บด้วยว่าบัญชีใดกำลังทำธุรกรรม และเมื่อเสร็จสิ้นให้เคลียร์ข้อมูลได้
    - ก่อนทำรายการต้องตรวจสอบว่า บัตรประชาชนของผู้มาทำรายการตรงกับเลขบัตรประชาชนของเจ้าของบัญชี
    - ไม่มีวงเงินการถอนรายวัน
- **เครื่องรูดบัตร (EDC Machine)**
    - รองรับเฉพาะ Debit Card
    - ต้องมีการ รูดบัตร และ ตรวจสอบ PIN
    - ใช้สำหรับชำระเงินให้ร้านค้า (Merchant)
    - Merchant ต้องมีบัญชีประเภท Current Account เท่านั้น
    - มีวงเงินรายวัน: 40,000 บาท
- **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], I=Interest/Cashback, P=Payment, F=Fee)
    - รูปแบบการบันทึก [Type]-[ChannelType]:[ChannelID]-[Amount]-[Balance]-[Target]
        - `D-ATM_machine:ATM-1001-5000.00-25000.00`
        - `W-Counter:COUNTER-01-50000.00-1000.00`
        - `P-EDC_machine:EDC-001-5000.00-24550.00-9000000001`

5. 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()) 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

6. **Skeleton Code:** ให้ดาวน์โหลดไฟล์ lab5_skeleton.py
    - ให้นักศึกษาเขียน Class Definition ให้ครบถ้วน
    - **ห้ามแก้ไข** Code ในส่วน create_bank_system() และ run_test() โดยเด็ดขาด
    - โปรแกรมที่เขียนต้องทำให้ Test Case ใน run_test() ทำงานได้ถูกต้อง

7. ข้อมูลจำลองสำหรับการทดสอบ (Test Data)
    - **User 1:** Harry Potter (Citizen ID: 1-1101-12345-12-0)
        - **บัญชีที่ 1: Saving Account** 1000000001, Balance: 20,000
            - **Card :** Premium Card (Debit Card) No. 12345 Pin : 1234
        - **บัญชีที่ 2: Fixed Account** 1000000002, Balance: 100,000 บาท
        - **บัญชีที่ 3: Current Account** 1000000003, Balance: 50,000 บาท
    - **User 2**: Hermione Granger (Citizen ID: 1-1101-12345-13-0)
        - **บัญชีที่ 1: Saving Account** 2000000001, Balance: 30,000
            - **Card :** Shopping Card (Debit Card) No. 22345 Pin : 5678
    - **User 3:** Merchant**: Shop ABC (Citizen ID:** 1-9999-99999-99-0)
        - **บัญชีที่ 1: Current Account** (สำหรับรับเงินผ่าน EDC) Account No: 9000000001 Balance: 100,000 บาท

    - **ช่องทางบริการ (Channels)**
        - **ATM Machine 1** ATM ID: ATM-1001 เงินสดในตู้: 1,000,000 บาท
        - **Counter 1 Counter** ID: COUNTER-01
        - **EDC Machine 1** EDC ID: EDC-001 Merchant Account: 9000000001 (Shop ABC)

8. **รายการทดสอบ (Test Cases)**

โปรแกรมต้องผ่านการทดสอบ ดังนี้:

<u>กลุ่มที่ 1 : ทดสอบเรื่องจำนวนเงิน 0 บาท เงินติดลบ เลข Pin และ ทำรายการโดยไม่เสียบบัตร</u>

- **Test Case #1:** ทดสอบไม่มีบัตรในเครื่อง โดยพยายามถอนเงินโดยไม่ใส่บัตรใน ATM
- **Test Case #2:** ทดสอบ Amount = 0 โดยพยายามฝากเงิน 0 บาท
- **Test Case #3:** ทดสอบ Amount = 0 สำหรับ EDC โดยพยายามจ่ายเงิน 0 บาทผ่าน EDC
- **Test Case #4:** ทดสอบ Amount ติดลบ โดยพยายามฝากเงิน –500 บาท
- **Test Case #5:** ทดสอบ PIN ผิด โดยใส่บัตร Harry ด้วย PIN = 9999 (ผิด)
- **Test Case #6:** ทดสอบ PIN ถูกต้อง โดยใส่บัตร Harry ด้วย PIN = 1234 (ถูก)

<u>กลุ่มที่ 2 : ทดสอบการฝากถอนโอนเงินผ่านช่องทางต่างๆ ทั้งสำเร็จและไม่สำเร็จ (ทดสอบเงื่อนไขการถอนสูงสุดต่อวัน และ เงื่อนไขต้องเหลือเงินไม่น้อยกว่าค่าธรรมเนียมรายปี) ทดสอบเงินไม่พอ</u>

- **Test Case #7:** ทดสอบการฝากเงินผ่าน ATM – สำเร็จ โดยใส่บัตร Harry และฝาก 5,000 บาท
- **Test Case #8:** ทดสอบการฝากเงินผ่าน Counter – สำเร็จ โดยยืนยันตัวตนด้วย Citizen ID และฝาก 10,000 บาท
- **Test Case #9:** ทดสอบ Saving Account Limit (40,000/ครั้ง) กับ Shopping Card โดยพยายามถอน 45,000 บาท
- **Test Case #10:** ทดสอบ Premium Card – ถอนได้ 40,000/ครั้ง โดยถอน 40,000 สำเร็จ แต่ถอน 50,000 ไม่สำเร็จ
- **Test Case #11:** ทดสอบ ATM – ต้องเหลือเงิน > ค่าธรรมเนียมรายปี โดยพยายามถอนจนเหลือน้อยกว่า 500 บาท
- **Test Case #12:** ทดสอบ Shopping Card – ถอนสำเร็จ โดยถอน 20,000 บาท (ภายใน limit)
- **Test Case #13:** ทดสอบการถอนเกินยอดในบัญชี โดยพยายามถอน 100,000 บาท
- **Test Case #14:** ทดสอบ ATM – เงินในตู้ไม่พอ โดยสร้าง ATM เงิน 500 บาท แล้วพยายามถอน 1,000 บาท
- **Test Case #15:** ทดสอบ Premium Card Daily Limit (100,000/วัน) โดยถอน 40,000 + 20,000 (รวม 100,000) แล้วพยายามถอนอีก 10,000
- **Test Case #16:** ทดสอบการโอนเงิน – ยอดไม่พอ โดยพยายามโอน 200,000 บาท
- **Test Case #17:** ทดสอบการโอนเงินผ่าน Counter โดยโอน 2,000 บาทผ่าน Counter

กลุ่มที่ 3 : ทดสอบเรื่องคำนวณดอกเบี้ย

- **Test Case #18:** ทดสอบ Saving Account – คำนวณดอกเบี้ย 0.5% โดยเรียก calculate_interest() ของ Harry's Saving
- **Test Case #19:** ทดสอบ Fixed Account – คำนวณดอกเบี้ย 2.5% โดยเรียก calculate_interest() ของ Harry's Fixed โดยสมมติว่าครบ 12 เดือนแล้ว
- **Test Case #20:** ทดสอบ Fixed Account – Early Withdrawal Warning โดยถอนเงินก่อนครบกำหนด 12 เดือน
- **Test Case #21:** ทดสอบ Current Account – ไม่มีดอกเบี้ย โดยเรียก calculate_interest() ของ Harry's Current
- **Test Case #22:** ทดสอบ Current Account – ถอนได้ไม่จำกัดต่อครั้ง โดยถอน 20,000 บาท ผ่าน Counter

กลุ่มที่ 4 : ทดสอบเรื่องการหักค่าธรรมเนียมบัตรและ cashback

- **Test Case #23:** ทดสอบ Annual Fee – ยอดเงินไม่พอ โดยสร้างบัญชียอด 100 บาท แล้วพยายามหัก Annual Fee 300 บาท
- **Test Case #24:** ทดสอบ ATM Card – Annual Fee 100 บาท โดยสร้างบัญชี + ATM Card แล้วหักค่าธรรมเนียม
- **Test Case #25:** ทดสอบ Premium Card – Annual Fee 500 บาท โดยหักค่าธรรมเนียมจาก Harry's Saving
- **Test Case #26:** ทดสอบ Premium Card – Cashback 2% (ไม่มีขั้นต่ำ) โดยจ่าย 5,000 บาท ผ่าน EDC
- **Test Case #27:** ทดสอบ Shopping Card – Cashback 1% (≥ 1,000) โดยจ่าย 3,000 บาท ผ่าน EDC

กลุ่มที่ 5 : ทดสอบความผิดพลาดต่างๆ เกี่ยวกับช่องทางการทำรายการ

- **Test Case #28:** ทดสอบ Counter – Citizen ID ผิด โดยใช้ Citizen ID = 0-0000-00000-00-0
- **Test Case #29:** ทดสอบ EDC – จ่ายเงินโดยไม่รูดบัตร โดยพยายาม pay() โดยไม่ swipe_card()
- **Test Case #30:** ทดสอบ EDC – จ่ายเงินเกินยอดในบัญชี โดยพยายามจ่าย 100,000 บาท

กลุ่มที่ 6 : ทดสอบความผิดพลาดของระบบ

- **Test Case #31:** ทดสอบ Type Validations โดยพยายาม add string แทนที่จะเป็น object ที่ถูกต้อง
- **Test Case #32:** ทดสอบ Search – หาบัตรที่ไม่มีอยู่ โดย search_account_from_card('99999')

- **Test Case #33:** ทดสอบ EDC Constructor – Merchant Account Type ผิด โดยสร้าง EDC ด้วย Saving Account

ส่วนที่ 2: ทดสอบ INHERITANCE

- **Test Case #34:** ทดสอบ Inheritance – All Accounts โดยใช้ isinstance() ตรวจสอบว่าทุก account เป็น instance ของ Account
- **Test Case #35:** ทดสอบ Inheritance – All Cards โดยตรวจสอบว่าทุกบัตรเป็น instance ของ Card
- **Test Case #36:** ทดสอบ Inheritance – All Channels โดยตรวจสอบว่าทุก channel เป็น instance ของ Channel
- **Test Case #37:** ทดสอบ Abstract Class – ไม่สามารถสร้าง Account instance โดยพยายาม Account('999', user, 1000)
- **Test Case #38:** ทดสอบ Abstract Class – ไม่สามารถสร้าง Card instance โดยพยายาม Card('999', '999', '9999')
- **Test Case #39:** ทดสอบ calculate_interest() แตกต่างกัน โดยเรียก method จากทุก account type
- **Test Case #40:** ทดสอบทุกบัตรจำกัด 40,000/ครั้ง โดยพยายามถอน > 40,000 จากทุกประเภทบัตร
- **Test Case #41:** ทดสอบ get_account_type() คืนค่าต่างกัน โดยเรียก method จากทุก account
- **Test Case #42:** ทดสอบ Card Types ต่างกัน โดยเรียก get_card_type() จากทุกบัตร
- **Test Case #43:** ทดสอบ Card Features – เปรียบเทียบ Premium vs Shopping vs ATM โดยแสดงตารางเปรียบเทียบ
- **Test Case #44:** ทดสอบ ประมวลผล Mixed Types โดยวนลูปผ่าน list ของ accounts ผสมกัน

การตรวจ

1. เมื่อเขียน Class Diagram เสร็จ ให้ Staff ตรวจ โดยให้ตอบคำถาม เรื่อง ความสัมพันธ์ระหว่าง Class จาก Class Diagram ที่นักศึกษาออกแบบ โดยเน้นเรื่อง Inheritance
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
- Method hasattr ใช้ในการตรวจสอบว่าใน object นั้นมี attribute นั้นหรือไม่