การเขียนโค้ดและการทดสอบที่ได้รับการช่วยเหลือจาก AI: การประเมินประสิทธิภาพและคุณภาพของโค้ด

AI-Assisted Coding and Testing: A Performance and Code Quality Evaluation

จัดทำโดย

นาย กัมแพงเพชร สิงห์ขรณ์ รหัสนักศึกษา 653380120-2

นาย ณภัทร ประสงค์ดี รหัสนักศึกษา 653380128-6

นาย ธีธัช เจริญวารี รหัสนักศึกษา 653380268-0

อาจารย์ประจำวิชา

ผศ. ดร.ชิตสุธา สุ่มเล็ก

เอกสารนี้เป็นส่วนหนึ่งของวิชา การประกันคุณภาพซอฟต์แวร์

(Software Quality Assurance)

ประจำภาคเรียนที่ 1 ปีการศึกษา 2567

วิทยาลัยการคอมพิวเตอร์ มหาวิทยาลัยขอนแก่น

ชื่อหัวข้อโครงงานภาษาไทย : การเขียนโค้ดและการทดสอบที่ได้รับการช่วยเหลือจาก AI: การประเมินประสิทธิภาพและคุณภาพของโค้ด

ชื่อหัวข้อโครงงานภาษาอังกฤษ : AI-Assisted Coding and Testing: A Performance and Code Quality Evaluation

ชื่อผู้จัดทำโครงงาน : นาย กัมแพงเพชร สิงห์ขรณ์ รหัสนักศึกษา 653380120-2
นาย ณภัทร ประสงค์ดี รหัสนักศึกษา 653380128-6
นาย ธีธัช เจริญวารี รหัสนักศึกษา 653380268-0

อาจารย์ประจำวิชา : ผศ. ดร.ชิตสุธา สุ่มเล็ก

สาขาวิชา : วิทยาการคอมพิวเตอร์ คณะวิทยาลัยการคอมพิวเตอร์

ปีการศึกษา : 2567

บทคัดย่อ

ในยุคปัจจุบัน AI กำลังเข้ามามีบทบาทสำคัญในวงการพัฒนาซอฟต์แวร์ โดยเฉพาะอย่างยิ่งในด้านการเขียนโค้ดและการทดสอบ ซึ่งเป็นที่คาดหวังว่าจะช่วยเพิ่มทั้งประสิทธิภาพและคุณภาพของซอฟต์แวร์ได้อย่างมาก

ผลงานวิจัยชิ้นนี้จึงมุ่งเน้นไปที่การศึกษาผลกระทบของ AI ต่อกระบวนการพัฒนาซอฟต์แวร์ โดยเปรียบเทียบโค้ดที่เขียนด้วย ChatGPT4o, Gemini 1.5 Pro, Gemini 1.5 Flash, GitHub Copilot เพื่อวิเคราะห์ประสิทธิภาพและคุณภาพแต่ละ Generative AI

การวิจัยครอบคลุมการตรวจสอบคุณภาพของโค้ดในหลายมิติ เช่น การหาข้อผิดพลาด การปฏิบัติตามมาตรฐานการเขียนโค้ด และการประเมินประสิทธิภาพการทำงานของโปรแกรม ผลการวิจัยจะช่วยให้เราเข้าใจถึงข้อดีและข้อจำกัดของการใช้ Generative AI ในการพัฒนาและทดสอบซอฟต์แวร์ ทั้งในแง่ของความเร็ว ความถูกต้อง และคุณภาพโดยรวม

ข้อมูลเชิงลึกจากงานวิจัยนี้จะเป็นประโยชน์อย่างยิ่งต่อผู้พัฒนาและผู้จัดการซอฟต์แวร์ ในการตัดสินใจเลือกใช้เทคโนโลยี Generative AI ในกระบวนการพัฒนาซอฟต์แวร์ได้อย่างเหมาะสมและมีประสิทธิภาพมากขึ้น

การทดลอง และ ผลลัพธ์

1. Chat-GPT 4o

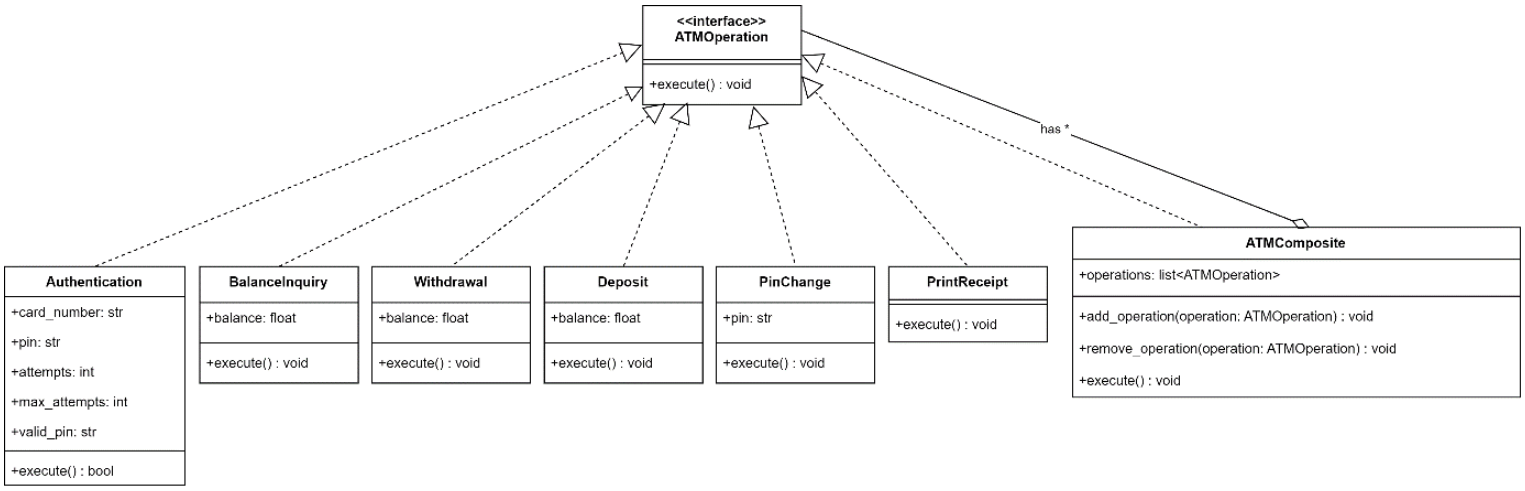
Prompt ที่ใช้ในการ Generate Code Python และ Generate Test ทั้งสามรอบ โดยใช้ Composite Design Pattern ในการ Gen Code

Use the composite design pattern to Write code in Python programming language from the requirements below.
"ความต้องการระบบตู้ ATM
1. การยืนยันตัวตนผู้ใช้
การป้อนข้อมูล: ผู้ใช้ต้องสามารถใส่บัตร ATM และรหัส PIN ได้
การตรวจสอบ: ระบบต้องตรวจสอบความถูกต้องของบัตรและรหัส PIN
การจำกัดความพยายาม: หากใส่ผิดเกินจำนวนครั้งที่กำหนด บัตรจะถูกยึด
2. การแสดงยอดเงินคงเหลือ
การเข้าถึงข้อมูล: ผู้ใช้สามารถเลือกดูยอดเงินคงเหลือในบัญชีได้
การแสดงผล: ระบบต้องแสดงยอดเงินคงเหลือปัจจุบัน
3. การถอนเงิน
การเลือกจำนวนเงิน: ผู้ใช้สามารถเลือกจำนวนเงินที่ต้องการถอนได้
การตรวจสอบยอดเงิน: ระบบต้องตรวจสอบว่ามียอดเงินเพียงพอสำหรับการถอน
การจ่ายเงิน: หากมียอดเงินเพียงพอ ระบบจะจ่ายเงินให้ผู้ใช้ และปรับปรุงยอดเงินคงเหลือ
การแจ้งเตือน: หากมียอดเงินไม่เพียงพอ ระบบจะแจ้งเตือนผู้ใช้
4. การฝากเงิน
การรับเงินสด: ผู้ใช้สามารถใส่เงินสดเข้าเครื่อง ATM ได้
การนับเงิน: ระบบต้องนับจำนวนเงินที่ฝากเข้ามา
การปรับปรุงยอดเงิน: ระบบต้องปรับปรุงยอดเงินคงเหลือในบัญชี
5. การเปลี่ยนรหัส PIN
การเข้าถึงฟังก์ชัน: ผู้ใช้สามารถเปลี่ยนรหัส PIN ของตนเองได้
การยืนยัน: ระบบต้องขอรหัส PIN เก่าและรหัส PIN ใหม่ 2 ครั้งเพื่อยืนยัน
การเปลี่ยนรหัส: หากรหัส PIN ใหม่ตรงกัน ระบบจะเปลี่ยนรหัส PIN ให้
6. การพิมพ์สลิป
การเลือกพิมพ์: หลังจากทำรายการเสร็จสิ้น ผู้ใช้สามารถเลือกพิมพ์สลิปได้
เนื้อหาสลิป: สลิปต้องแสดงข้อมูลรายละเอียดของรายการ เช่น วันที่ เวลา จำนวนเงิน และยอดเงินคงเหลือ"

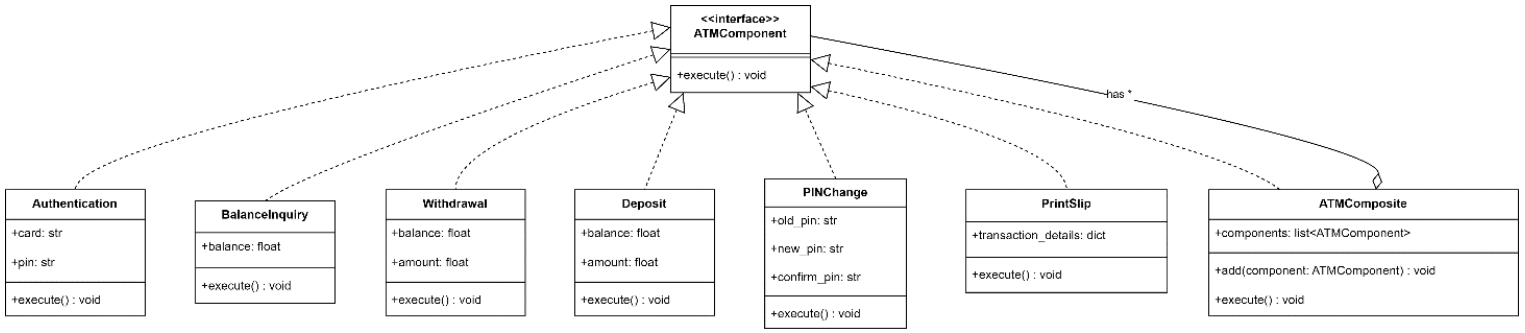
ใช้ Gen Test หลังจาก Gen code เสร็จ

Write a pytest to test your given code that has 100% statement coverage and 100% branch coverage.

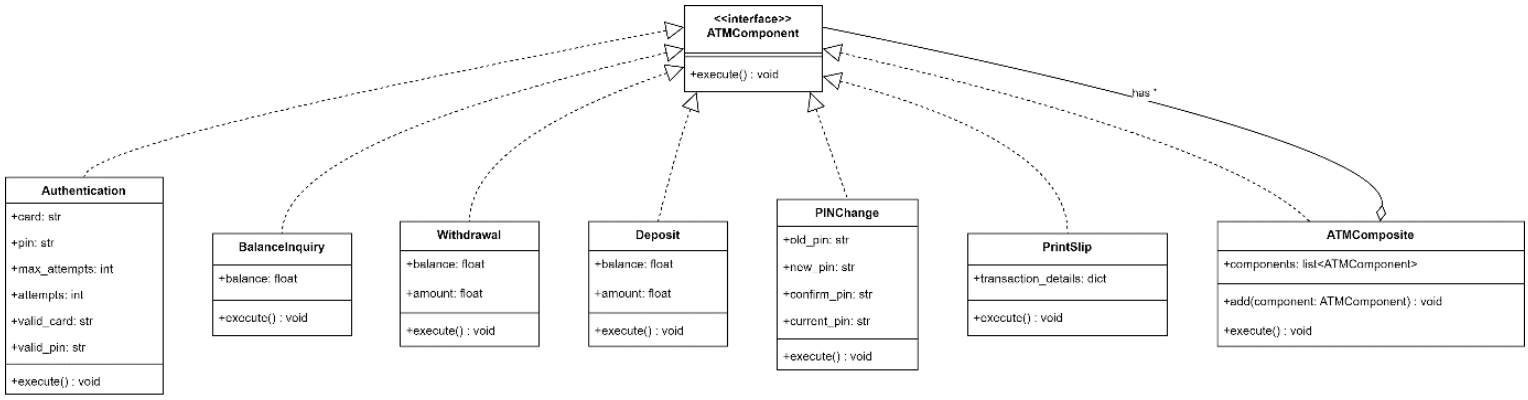
ผลลัพธ์ ในรอบที่ 1



ผลลัพธ์ ในรอบที่ 2



ผลลัพธ์ ในรอบที่ 3



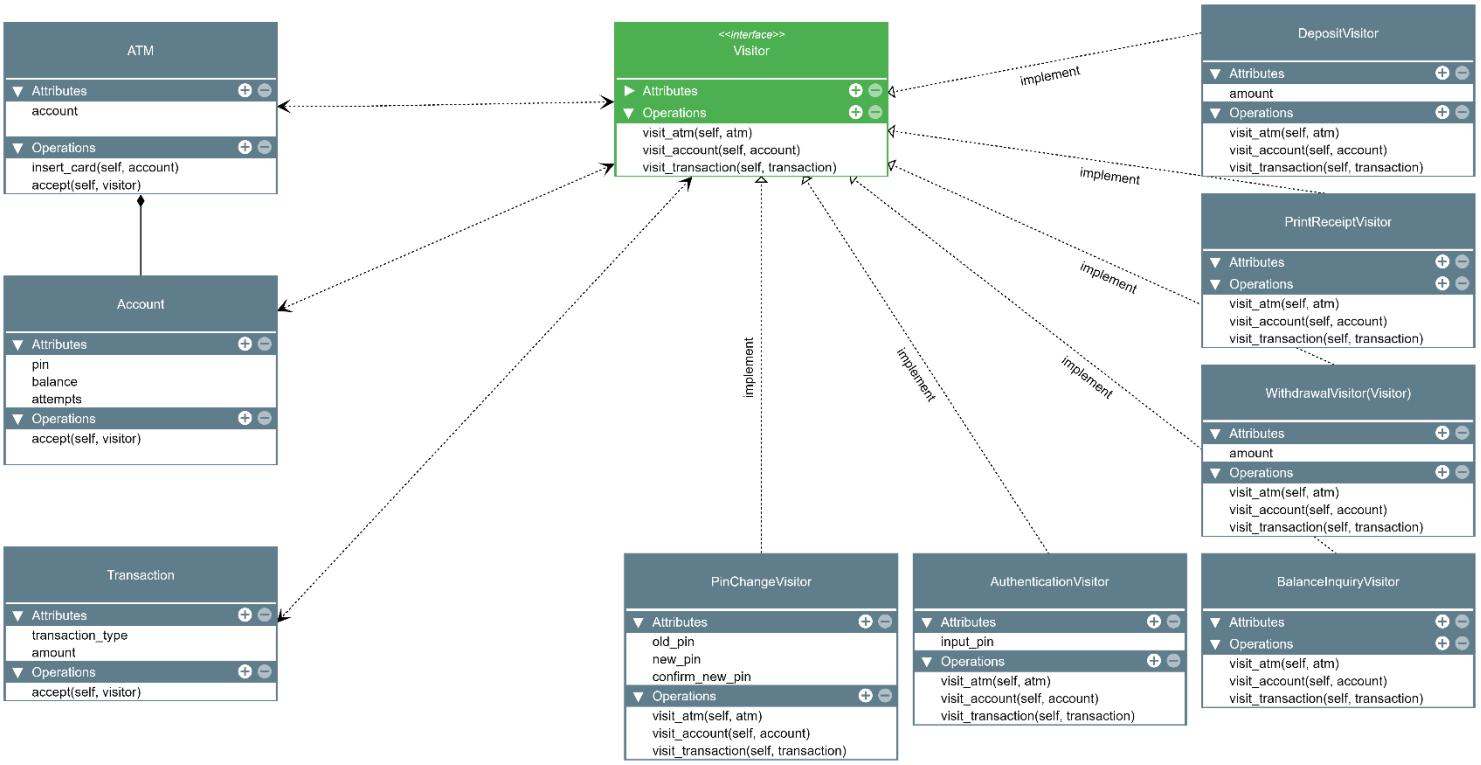
Prompt ที่ใช้ในการ Generate Code Python และ Generate Test ทั้งสามรอบ โดยใช้ Visitor Design Pattern ในการ Gen Code

Use the visitor design pattern to Write code in Python programming language from the requirements below.
"ความต้องการระบบตู้ ATM
1. การยืนยันตัวตนผู้ใช้
การป้อนข้อมูล: ผู้ใช้ต้องสามารถใส่บัตร ATM และรหัส PIN ได้
การตรวจสอบ: ระบบต้องตรวจสอบความถูกต้องของบัตรและรหัส PIN
การจำกัดความพยายาม: หากใส่ผิดเกินจำนวนครั้งที่กำหนด บัตรจะถูกยึด
2. การแสดงยอดเงินคงเหลือ
การเข้าถึงข้อมูล: ผู้ใช้สามารถเลือกดูยอดเงินคงเหลือในบัญชีได้
การแสดงผล: ระบบต้องแสดงยอดเงินคงเหลือปัจจุบัน
3. การถอนเงิน
การเลือกจำนวนเงิน: ผู้ใช้สามารถเลือกจำนวนเงินที่ต้องการถอนได้
การตรวจสอบยอดเงิน: ระบบต้องตรวจสอบว่ามียอดเงินเพียงพอสำหรับการถอน
การจ่ายเงิน: หากมียอดเงินเพียงพอ ระบบจะจ่ายเงินให้ผู้ใช้ และปรับปรุงยอดเงินคงเหลือ
การแจ้งเตือน: หากมียอดเงินไม่เพียงพอ ระบบจะแจ้งเตือนผู้ใช้
4. การฝากเงิน
การรับเงินสด: ผู้ใช้สามารถใส่เงินสดเข้าเครื่อง ATM ได้
การนับเงิน: ระบบต้องนับจำนวนเงินที่ฝากเข้ามา
การปรับปรุงยอดเงิน: ระบบต้องปรับปรุงยอดเงินคงเหลือในบัญชี
5. การเปลี่ยนรหัส PIN
การเข้าถึงฟังก์ชัน: ผู้ใช้สามารถเปลี่ยนรหัส PIN ของตนเองได้
การยืนยัน: ระบบต้องขอรหัส PIN เก่าและรหัส PIN ใหม่ 2 ครั้งเพื่อยืนยัน
การเปลี่ยนรหัส: หากรหัส PIN ใหม่ตรงกัน ระบบจะเปลี่ยนรหัส PIN ให้
6. การพิมพ์สลิป
การเลือกพิมพ์: หลังจากทำรายการเสร็จสิ้น ผู้ใช้สามารถเลือกพิมพ์สลิปได้
เนื้อหาสลิป: สลิปต้องแสดงข้อมูลรายละเอียดของรายการ เช่น วันที่ เวลา จำนวนเงิน และยอดเงินคงเหลือ"

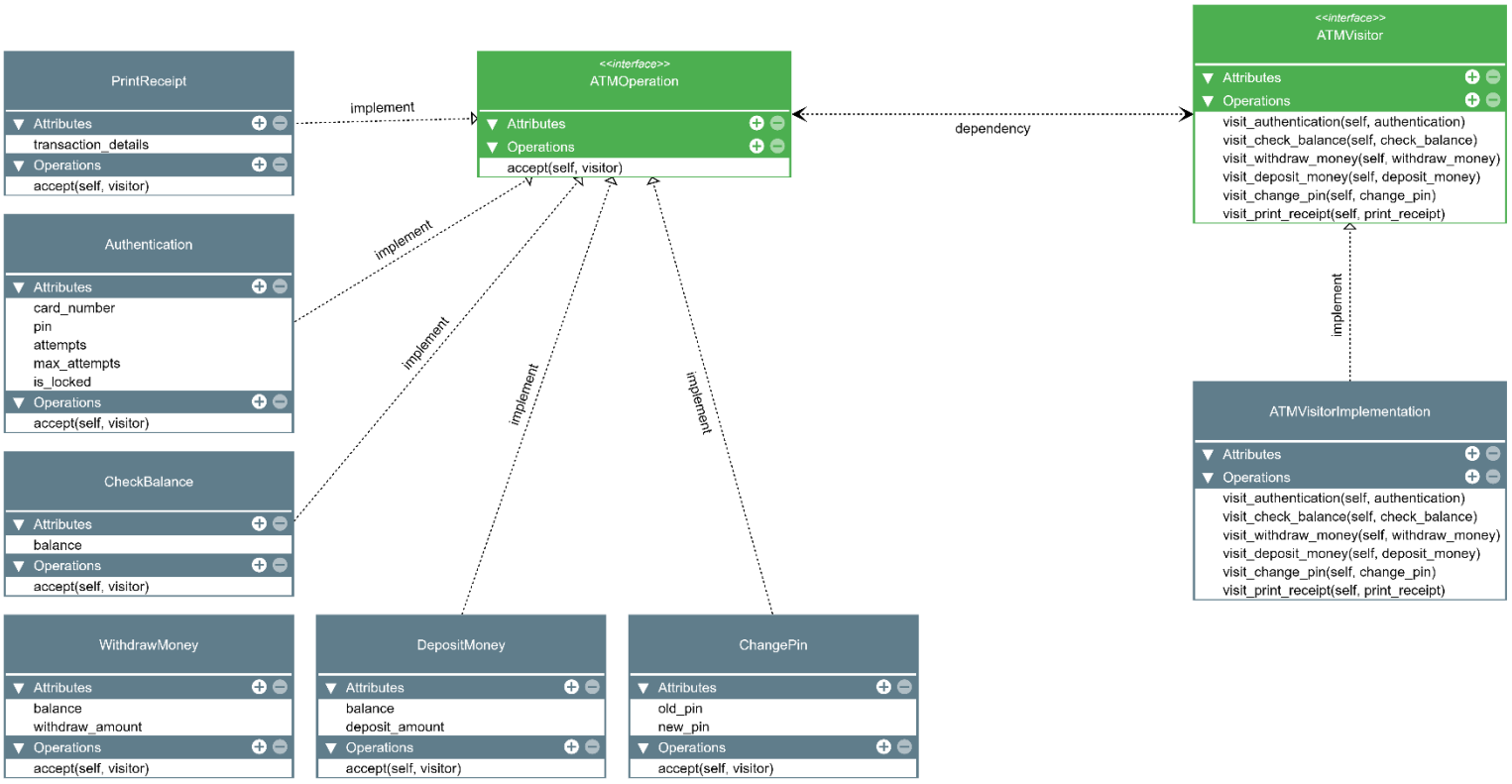
ใช้ GenTest หลังจาก Gen code เสร็จ

Write pytest to test your given code that has 100% statement coverage and 100% branch coverage.

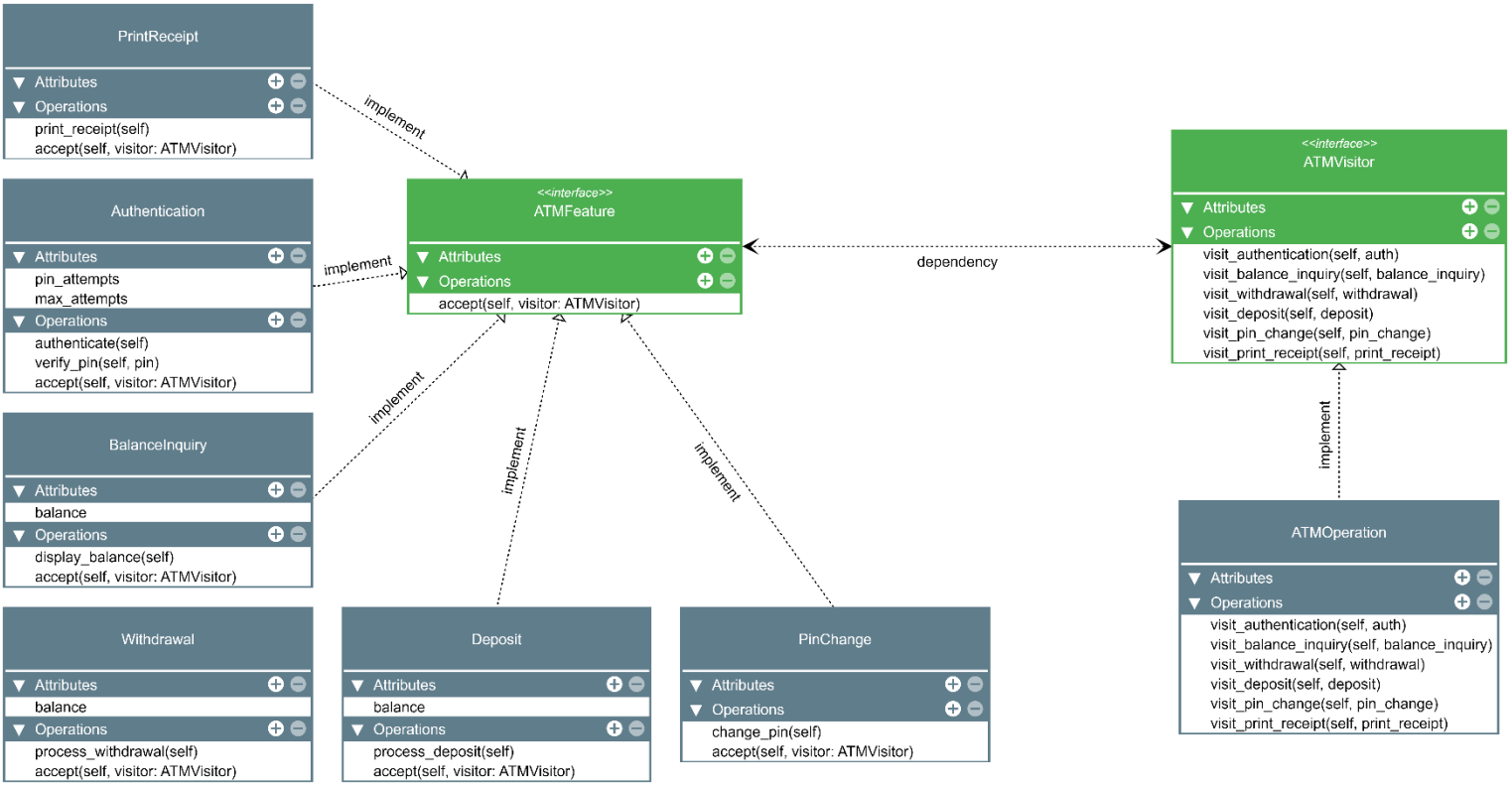
ผลลัพธ์ ในรอบที่ 1



ผลลัพธ์ ในรอบที่ 2



ผลลัพธ์ ในรอบที่ 3



วิเคราะห์ผลการทดลองการ Generate Code In Python Programing Language by Chat-GPT 4o

จากผลลัพธ์ พบว่า Chat-GPT 4o สามารถสร้างโค้ดถูกต้องตาม Requirement แต่รายละเอียดการทำงานของฟังก์ชันนั้นจะไม่ซับซ้อนเป็นเพียงแค่ตัวอย่างว่าเมื่อฟังก์ชันได้รับ input เข้ามาแล้วหรือถูกเรียกใช้ ฟังก์ชันจะตอบสนองและส่งคืนค่ากลับหรือแสดงผลอย่างไร ตาม Requirement ที่ระบุ

Chat-GPT 4o สร้างโค้ดขึ้นโดยใช้ภาษาโปรแกรมและ Design pattern ที่ระบุได้ถูกต้องทั้ง Composite Design Pattern และ Visitor Design แต่มีผิดพลาดอยู่ 1 ผลลัพธ์ นั่นคือ

ผลลัพธ์ในรอบที่ 1 Visitor Design Pattern ที่สร้างโค้ดผิดจากข้อกำหนดของ Visitor Design Pattern เนื่องจากไม่มีการสร้าง interface Element แต่เป็นการสร้าง Concrete Element Class (ATM Account Transaction) ขึ้นมาเรียกใช้ Visitor โดยตรง ซึ่งถือว่าผิดจากข้อกำหนดของ Visitor Design Pattern

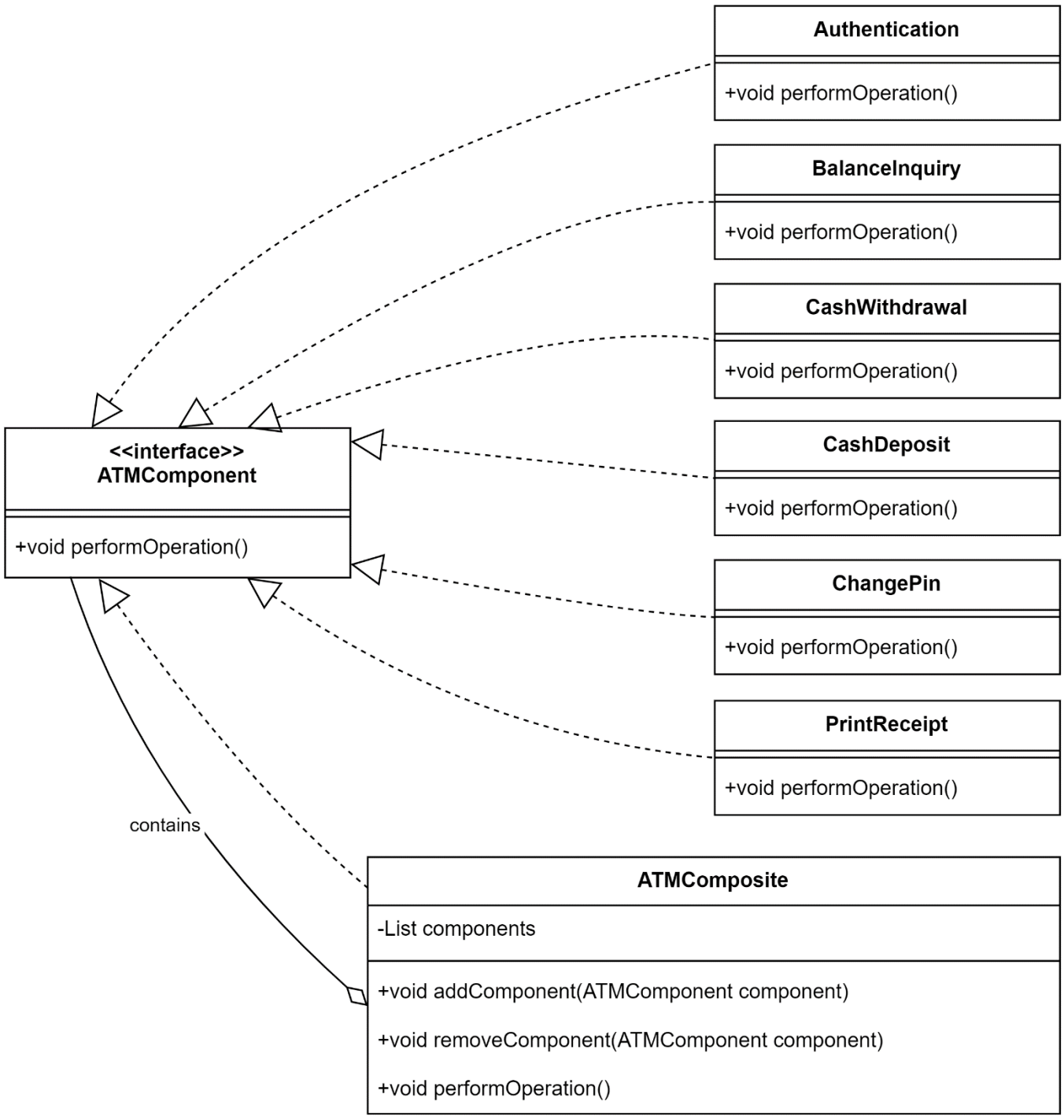
Prompt ที่ใช้ในการ Generate Code Java และ Generate Test ทั้งสามรอบ โดยใช้ Composite Design Pattern ในการ Gen Code

Use the composite design pattern to Write code in Java programming language from the requirements below.
"ความต้องการระบบตู้ ATM
1. การยืนยันตัวตนผู้ใช้
การป้อนข้อมูล: ผู้ใช้ต้องสามารถใส่บัตร ATM และรหัส PIN ได้
การตรวจสอบ: ระบบต้องตรวจสอบความถูกต้องของบัตรและรหัส PIN
การจำกัดความพยายาม: หากใส่ผิดเกินจำนวนครั้งที่กำหนด บัตรจะถูกยึด
2. การแสดงยอดเงินคงเหลือ
การเข้าถึงข้อมูล: ผู้ใช้สามารถเลือกดูยอดเงินคงเหลือในบัญชีได้
การแสดงผล: ระบบต้องแสดงยอดเงินคงเหลือปัจจุบัน
3. การถอนเงิน
การเลือกจำนวนเงิน: ผู้ใช้สามารถเลือกจำนวนเงินที่ต้องการถอนได้
การตรวจสอบยอดเงิน: ระบบต้องตรวจสอบว่ามียอดเงินเพียงพอสำหรับการถอน
การจ่ายเงิน: หากมียอดเงินเพียงพอ ระบบจะจ่ายเงินให้ผู้ใช้ และปรับปรุงยอดเงินคงเหลือ
การแจ้งเตือน: หากมียอดเงินไม่เพียงพอ ระบบจะแจ้งเตือนผู้ใช้
4. การฝากเงิน
การรับเงินสด: ผู้ใช้สามารถใส่เงินสดเข้าเครื่อง ATM ได้
การนับเงิน: ระบบต้องนับจำนวนเงินที่ฝากเข้ามา
การปรับปรุงยอดเงิน: ระบบต้องปรับปรุงยอดเงินคงเหลือในบัญชี
5. การเปลี่ยนรหัส PIN
การเข้าถึงฟังก์ชัน: ผู้ใช้สามารถเปลี่ยนรหัส PIN ของตนเองได้
การยืนยัน: ระบบต้องขอรหัส PIN เก่าและรหัส PIN ใหม่ 2 ครั้งเพื่อยืนยัน
การเปลี่ยนรหัส: หากรหัส PIN ใหม่ตรงกัน ระบบจะเปลี่ยนรหัส PIN ให้
6. การพิมพ์สลิป
การเลือกพิมพ์: หลังจากทำรายการเสร็จสิ้น ผู้ใช้สามารถเลือกพิมพ์สลิปได้
เนื้อหาสลิป: สลิปต้องแสดงข้อมูลรายละเอียดของรายการ เช่น วันที่ เวลา จำนวนเงิน และยอดเงินคงเหลือ"

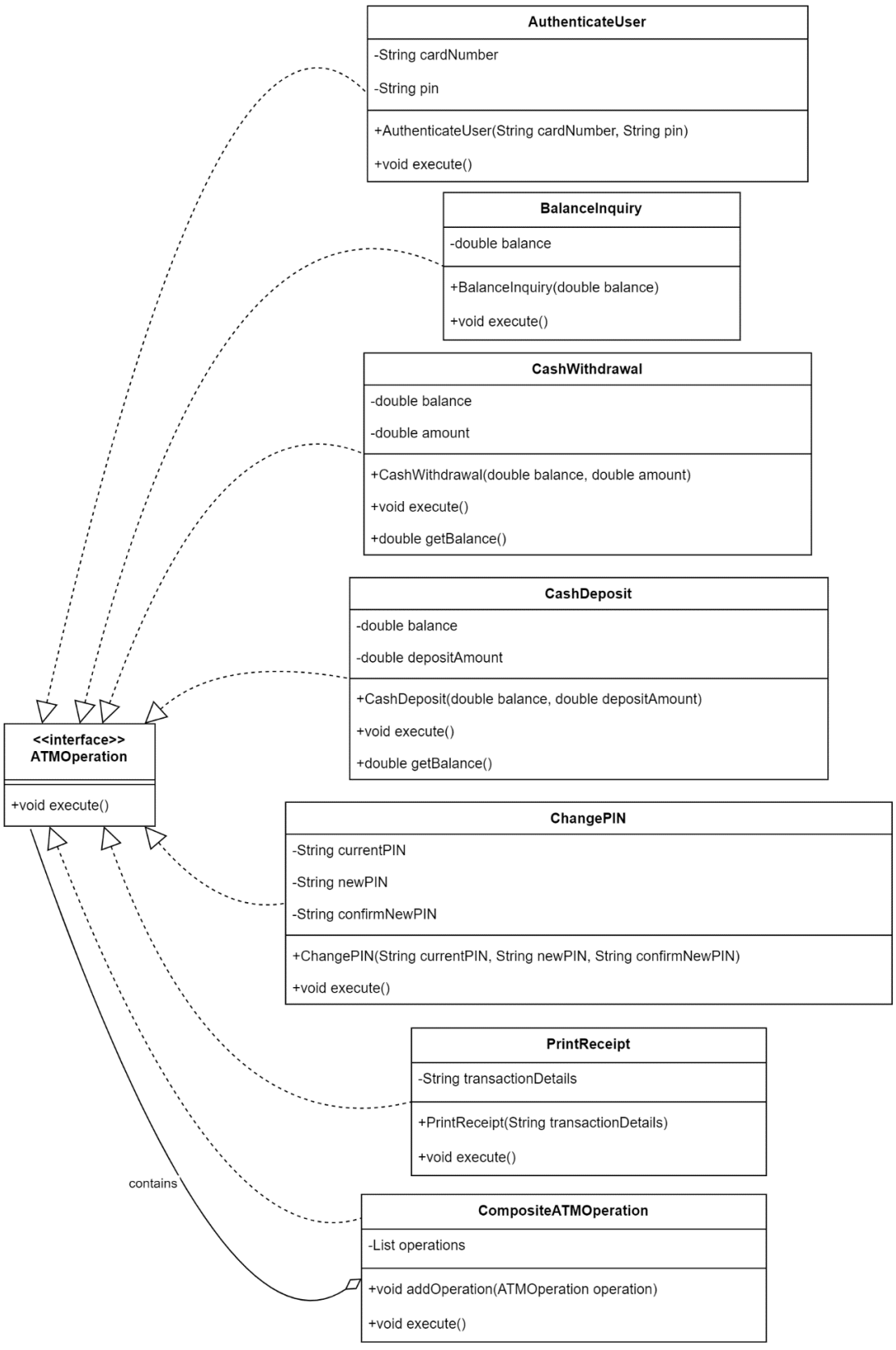
ใช้ GenTest หลังจาก Gen code เสร็จ

Write a JUnit to test your given code that has 100% statement coverage and 100% branch coverage.

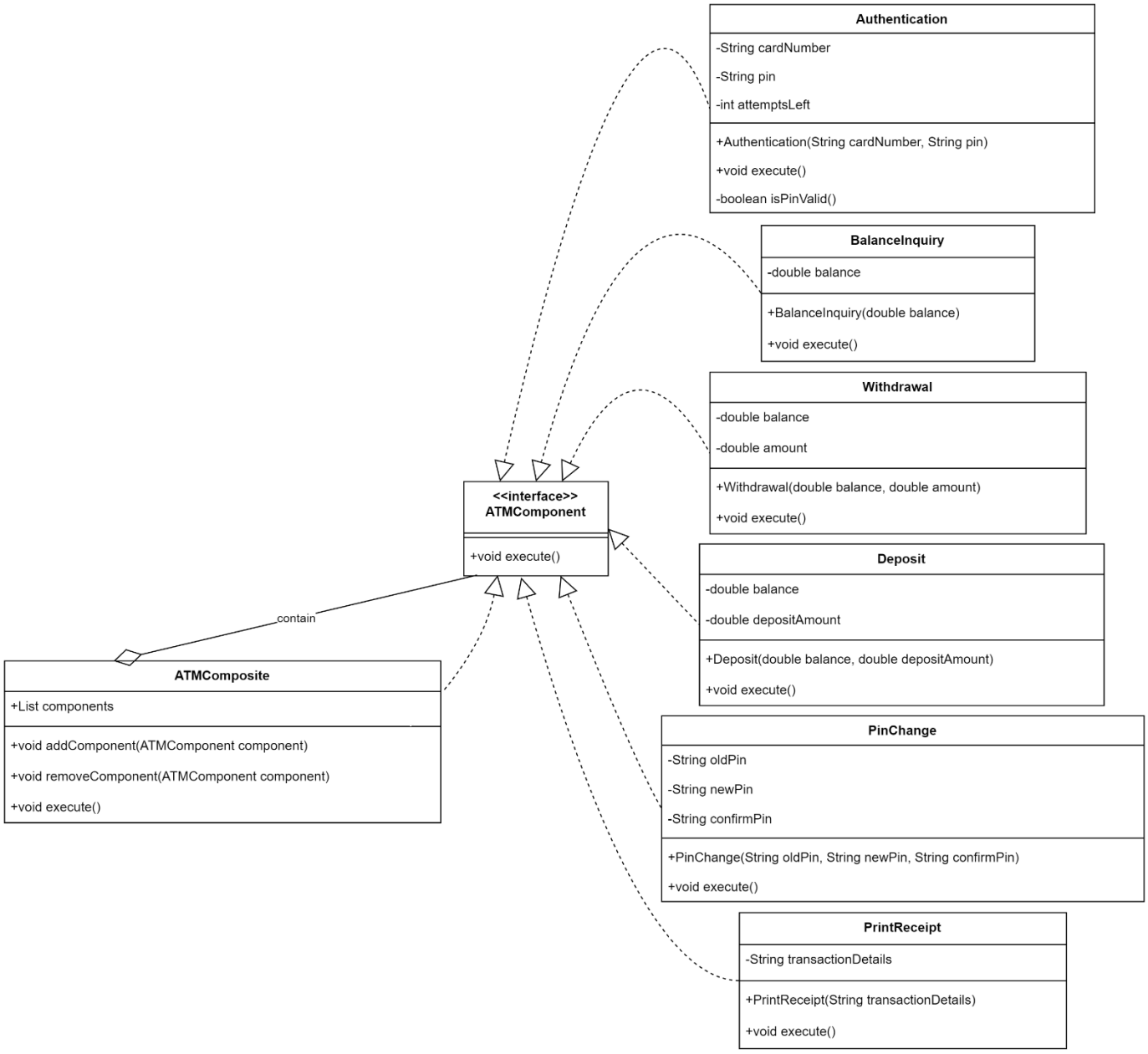
ผลลพธ์ในรอบที่ 1



ผลลพธ์ในรอบที่ 2



ผลลพธ์ในรอบที่ 3



Prompt ที่ใช้ในการ Generate Code Java และ Generate Test ทั้งสามรอบ โดยใช้ Visitor Design Pattern ในการ Gen Code

Use the visitor design pattern to Write code in Java programming language from the requirements below.

"ความต้องการระบบตู้ ATM

1. การยืนยันตัวตนผู้ใช้

การป้อนข้อมูล: ผู้ใช้ต้องสามารถใส่บัตร ATM และรหัส PIN ได้

การตรวจสอบ: ระบบต้องตรวจสอบความถูกต้องของบัตรและรหัส PIN

การจำกัดความพยายาม: หากใส่ผิดเกินจำนวนครั้งที่กำหนด บัตรจะถูกยึด

2. การแสดงยอดเงินคงเหลือ

การเข้าถึงข้อมูล: ผู้ใช้สามารถเลือกดูยอดเงินคงเหลือในบัญชีได้

การแสดงผล: ระบบต้องแสดงยอดเงินคงเหลือปัจจุบัน

3. การถอนเงิน

การเลือกจำนวนเงิน: ผู้ใช้สามารถเลือกจำนวนเงินที่ต้องการถอนได้

การตรวจสอบยอดเงิน: ระบบต้องตรวจสอบว่ามียอดเงินเพียงพอสำหรับการถอน

การจ่ายเงิน: หากมียอดเงินเพียงพอ ระบบจะจ่ายเงินให้ผู้ใช้ และปรับปรุงยอดเงินคงเหลือ

การแจ้งเตือน: หากมียอดเงินไม่เพียงพอ ระบบจะแจ้งเตือนผู้ใช้

4. การฝากเงิน

การรับเงินสด: ผู้ใช้สามารถใส่เงินสดเข้าเครื่อง ATM ได้

การนับเงิน: ระบบต้องนับจำนวนเงินที่ฝากเข้ามา

การปรับปรุงยอดเงิน: ระบบต้องปรับปรุงยอดเงินคงเหลือในบัญชี

5. การเปลี่ยนรหัส PIN

การเข้าถึงฟังก์ชัน: ผู้ใช้สามารถเปลี่ยนรหัส PIN ของตนเองได้

การยืนยัน: ระบบต้องขอรหัส PIN เก่าและรหัส PIN ใหม่ 2 ครั้งเพื่อยืนยัน

การเปลี่ยนรหัส: หากรหัส PIN ใหม่ตรงกัน ระบบจะเปลี่ยนรหัส PIN ให้

6. การพิมพ์สลิป

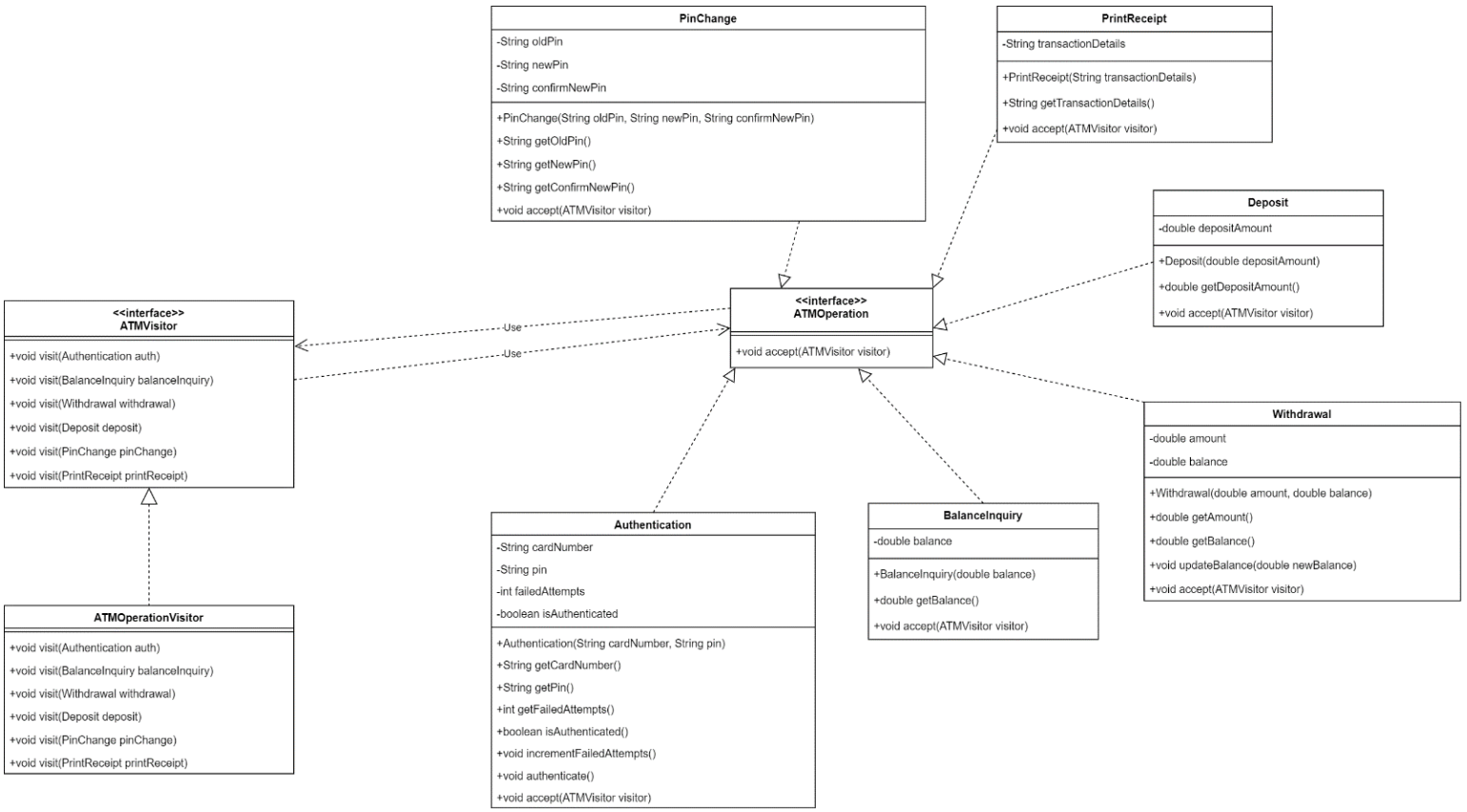
การเลือกพิมพ์: หลังจากทำรายการเสร็จสิ้น ผู้ใช้สามารถเลือกพิมพ์สลิปได้

เนื้อหาสลิป: สลิปต้องแสดงข้อมูลรายละเอียดของรายการ เช่น วันที่ เวลา จำนวนเงิน และยอดเงินคงเหลือ"

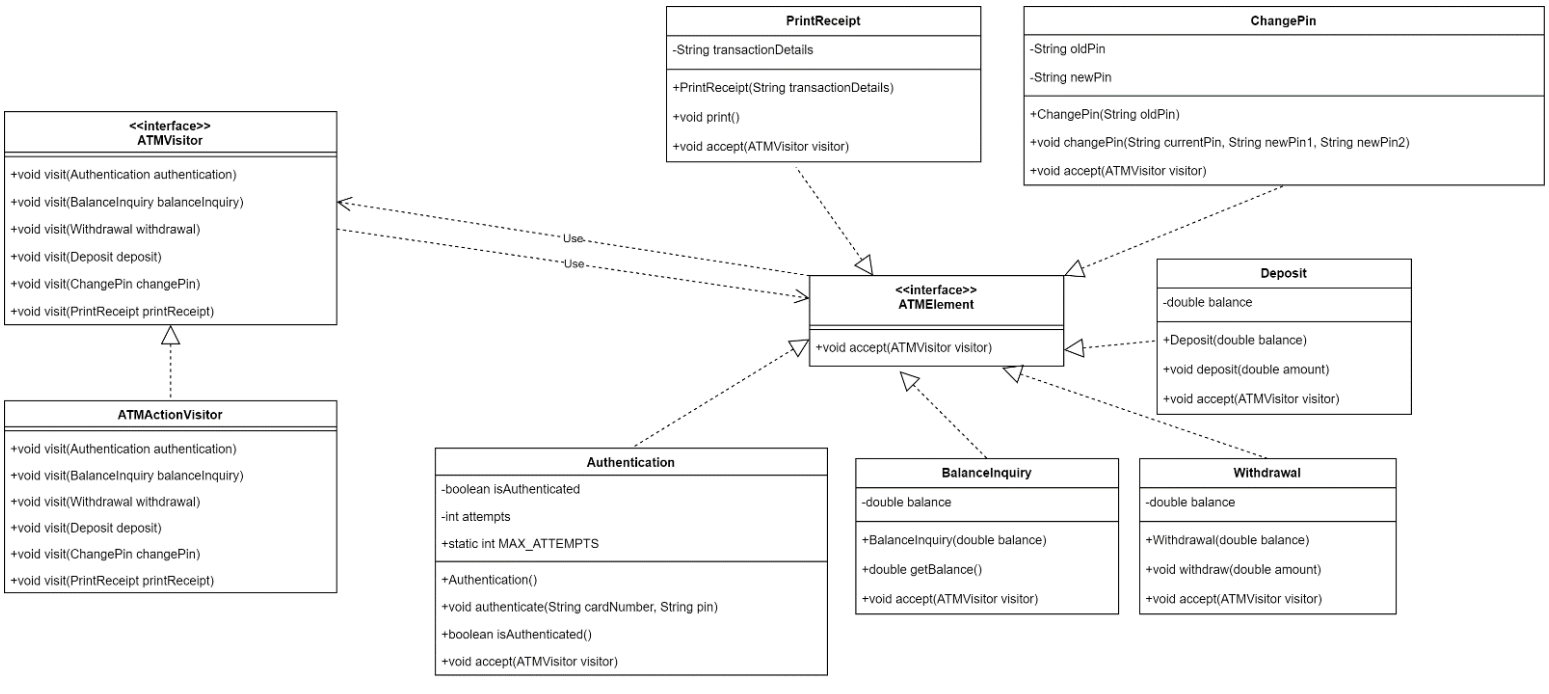
ใช้ GenTest หลังจาก Gen code เสร็จ

Write a JUnit to test your given code that has 100% statement coverage and 100% branch coverage.

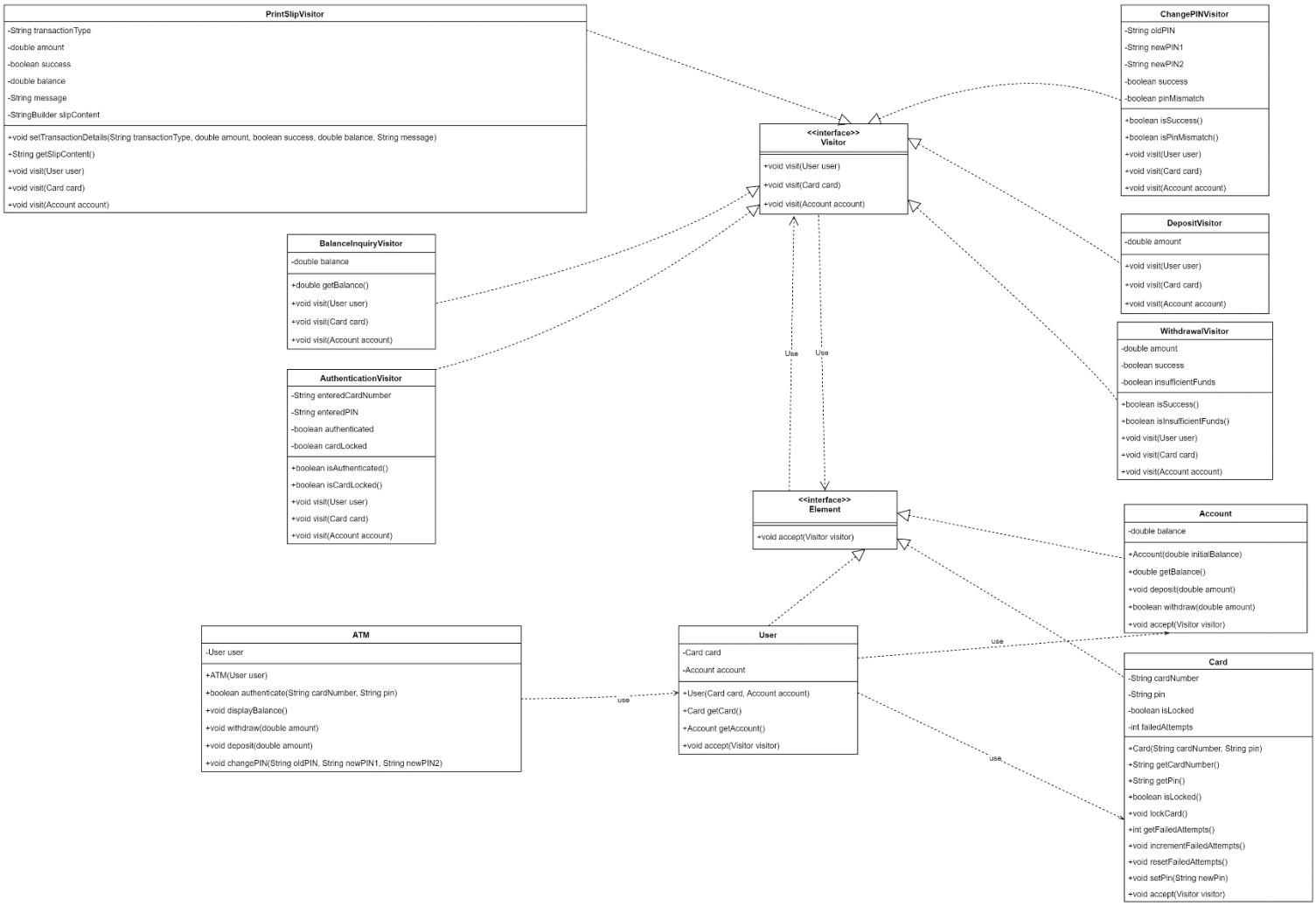
ผลลพธ์ในรอบที่ 1



ผลลพธ์ในรอบที่ 2



ผลลพธ์ในรอบที่ 3



วิเคราะห์ผลการทดลอง Generate Code In Java Programing Language by Chat-GPT 4o

จากผลลัพธ์ พบว่า Chat-GPT 4o สามารถสร้างโค้ดถูกต้องตาม Requirement แต่รายละเอียดการทำงานของฟังก์ชันนั้นจะไม่ซับซ้อนเป็นเพียงแค่ตัวอย่างว่าเมื่อฟังก์ชันได้รับ input เข้ามาแล้วหรือถูกเรียกใช้ ฟังก์ชันจะตอบสนองและส่งคืนค่ากลับหรือแสดงผลอย่างไร ตาม Requirement ที่ระบุ

Chat-GPT 4o สร้างโค้ดขึ้นโดยใช้ภาษาโปรแกรมและ Design pattern ที่ระบุได้ถูกต้องทั้ง Composite Design Pattern และ Visitor Design

มีผลลัพธ์ในรอบที่ 3 Visitor Design Pattern ที่รูปแบบโค้ดดูซับซ้อนเพิ่มขึ้น

สรุปผลการทดลอง Generate Code by Chat-GPT 4o ทั้ง ภาษา Python และ Java

จากการทดลองส่วนใหญ่พบว่า Chat-GPT สามารถช่วยออกแบบระบบตาม Requirement และ Design Patern ได้จริงแต่การทำงานของระบบจะไม่ซับซ้อนมากซึ่งในการ Generate Code ครั้งแรกๆนั้นอาจมีจุดผิดพลาดบางส่วนในการออกแบบตาม Design Patern ที่กำหนดไว้ และ ความซับซ้อนของฟังก์ชันในการทำงานในระบบน้อยไป แต่ในรอบการทดลองครั้งถัดมาก็มีการแก้ไขให้สมบูรณ์ขึ้น

โดยสรุป Chat-GPT 4o สามารถเป็นผู้ช่วยในการสร้าง ออกแบบ หรือ นำร่องการพัฒนาโปรแกรมได้ในระดับหนึ่ง ซึ่งนักพัฒนานั้นควรจะมีทักษะเฉพาะทาง และตรวจสอบผลลัพธ์ที่ได้จาก Gennerative AI Chat-GPT อย่างละเอียด ก่อนนำไปพัฒนาต่อ หรือใช้งานจริงเพื่อให้เกิดความถูกต้องสมบูรณ์ของระบบ

2. Gemini 1.5 Flash / Gemini 1.5 Pro

Prompt ที่ใช้ในการ Generate Code Python และ Generate Test ทั้งสามรอบ โดยใช้ Composite Design Pattern

We are developing a fully functional ATM system using the Python Programming Language and implementing the Composite Design Pattern. The system should meet the following requirements:

#System Requirements:

1)User Authentication:

1.1)Input: The system must allow users to insert their ATM card and enter a PIN.

1.2)Verification: The system must verify the correctness of the card and PIN.

1.3)Attempt Limitation: If the user enters the wrong PIN more than the allowed number of times, the card should be retained.

2)Balance Display:

2.1)Data Access: The user can view their account balance.

2.2)Display: The system must display the current balance.

3)Cash Withdrawal:

3.1)Amount Selection: The user can select the amount they wish to withdraw.

3.2)Balance Verification: The system must verify if there are sufficient funds for the withdrawal.

3.3)Dispensing Cash: If sufficient funds are available, the system will dispense the cash and update the balance.

3.4)Insufficient Funds Alert: If there are insufficient funds, the system will notify the user.

4)Deposit:

4.1)Cash Reception: The user can deposit cash into the ATM.

4.2)Money Counting: The system must count the deposited amount.

4.3)Balance Update: The system must update the account balance accordingly.

5)PIN Change:

5.1)Function Access: The user can change their PIN.

5.2)Confirmation: The system must request the old PIN and the new PIN (entered twice) for confirmation.

5.3)PIN Update: If the new PIN matches the confirmation, the system will update the user's PIN.

6)Receipt Printing:

6.1)Print Option: After completing a transaction, the user can choose to print a receipt.

6.2)Receipt Content: The receipt must show transaction details, such as date, time, amount, and the remaining balance.

#Testing Requirements:

Testing with pytest:

Develop unit tests to achieve 100% statement coverage and 100% branch coverage.

#Deliverables:

code.py: The full implementation of the ATM system.

unit_test.py: The pytest script with complete coverage tests.

Prompt ที่ใช้ในการ Generate Code Python และ Generate Test ทั้งสามรอบ โดยใช้ Visitor Design Pattern

We are developing a fully functional ATM system using the Python Programming Language and implementing the Visitor Design Pattern. The system should meet the following requirements:

#System Requirements:

1)User Authentication:

1.1)Input: The system must allow users to insert their ATM card and enter a PIN.

1.2)Verification: The system must verify the correctness of the card and PIN.

1.3)Attempt Limitation: If the user enters the wrong PIN more than the allowed number of times, the card should be retained.

2)Balance Display:

2.1)Data Access: The user can view their account balance.

2.2)Display: The system must display the current balance.

3)Cash Withdrawal:

3.1)Amount Selection: The user can select the amount they wish to withdraw.

3.2)Balance Verification: The system must verify if there are sufficient funds for the withdrawal.

3.3)Dispensing Cash: If sufficient funds are available, the system will dispense the cash and update the balance.

3.4)Insufficient Funds Alert: If there are insufficient funds, the system will notify the user.

4)Deposit:

4.1)Cash Reception: The user can deposit cash into the ATM.

4.2)Money Counting: The system must count the deposited amount.

4.3)Balance Update: The system must update the account balance accordingly.

5)PIN Change:

5.1)Function Access: The user can change their PIN.

5.2)Confirmation: The system must request the old PIN and the new PIN (entered twice) for confirmation.

5.3)PIN Update: If the new PIN matches the confirmation, the system will update the user's PIN.

6)Receipt Printing:

6.1)Print Option: After completing a transaction, the user can choose to print a receipt.

6.2)Receipt Content: The receipt must show transaction details, such as date, time, amount, and the remaining balance.

#Testing Requirements:

Testing with pytest:

Develop unit tests to achieve 100% statement coverage and 100% branch coverage.

#Deliverables:

code.py: The full implementation of the ATM system.

unit_test.py: The pytest script with complete coverage tests.

Prompt ที่ใช้ในการ Generate Code Java และ Generate Test ทั้งสามรอบ โดยใช้ Composite Design Pattern

We are developing a fully functional ATM system using the Java Programming Language and implementing the Composite Design Pattern. The system should meet the following requirements:

#System Requirements:

1)User Authentication:

1.1)Input: The system must allow users to insert their ATM card and enter a PIN.

1.2)Verification: The system must verify the correctness of the card and PIN.

1.3)Attempt Limitation: If the user enters the wrong PIN more than the allowed number of times, the card should be retained.

2)Balance Display:

2.1)Data Access: The user can view their account balance.

2.2)Display: The system must display the current balance.

3)Cash Withdrawal:

3.1)Amount Selection: The user can select the amount they wish to withdraw.

3.2)Balance Verification: The system must verify if there are sufficient funds for the withdrawal.

3.3)Dispensing Cash: If sufficient funds are available, the system will dispense the cash and update the balance.

3.4)Insufficient Funds Alert: If there are insufficient funds, the system will notify the user.

4)Deposit:

4.1)Cash Reception: The user can deposit cash into the ATM.

4.2)Money Counting: The system must count the deposited amount.

4.3)Balance Update: The system must update the account balance accordingly.

5)PIN Change:

5.1)Function Access: The user can change their PIN.

5.2)Confirmation: The system must request the old PIN and the new PIN (entered twice) for confirmation.

5.3)PIN Update: If the new PIN matches the confirmation, the system will update the user's PIN.

6)Receipt Printing:

6.1)Print Option: After completing a transaction, the user can choose to print a receipt.

6.2)Receipt Content: The receipt must show transaction details, such as date, time, amount, and the remaining balance.

#Testing Requirements:

Testing with JUnit5:

Develop unit tests to achieve 100% statement coverage and 100% branch coverage.

#Deliverables:

code : The full implementation of the ATM system.

unit_test : The JUnit5 script with complete coverage tests.

Prompt ที่ใช้ในการ Generate Code Java และ Generate Test ทั้งสามรอบ โดยใช้ Visitor Design Pattern

We are developing a fully functional ATM system using the Java Programming Language and implementing the Visitor Design Pattern. The system should meet the following requirements:

#System Requirements:

1)User Authentication:

1.1)Input: The system must allow users to insert their ATM card and enter a PIN.

1.2)Verification: The system must verify the correctness of the card and PIN.

1.3)Attempt Limitation: If the user enters the wrong PIN more than the allowed number of times, the card should be retained.

2)Balance Display:

2.1)Data Access: The user can view their account balance.

2.2)Display: The system must display the current balance.

3)Cash Withdrawal:

3.1)Amount Selection: The user can select the amount they wish to withdraw.

3.2)Balance Verification: The system must verify if there are sufficient funds for the withdrawal.

3.3)Dispensing Cash: If sufficient funds are available, the system will dispense the cash and update the balance.

3.4)Insufficient Funds Alert: If there are insufficient funds, the system will notify the user.

4)Deposit:

4.1)Cash Reception: The user can deposit cash into the ATM.

4.2)Money Counting: The system must count the deposited amount.

4.3)Balance Update: The system must update the account balance accordingly.

5)PIN Change:

5.1)Function Access: The user can change their PIN.

5.2)Confirmation: The system must request the old PIN and the new PIN (entered twice) for confirmation.

5.3)PIN Update: If the new PIN matches the confirmation, the system will update the user's PIN.

6)Receipt Printing:

6.1)Print Option: After completing a transaction, the user can choose to print a receipt.

6.2)Receipt Content: The receipt must show transaction details, such as date, time, amount, and the remaining balance.

#Testing Requirements:

Testing with JUnit5:

Develop unit tests to achieve 100% statement coverage and 100% branch coverage.

#Deliverables:

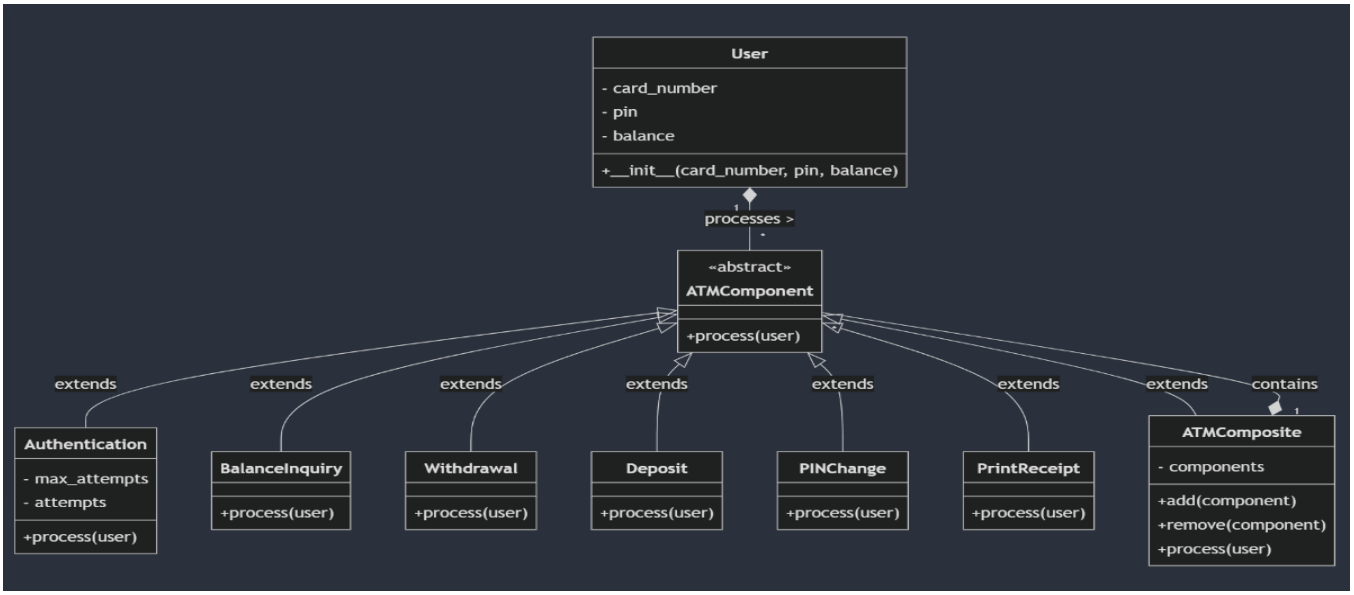
code : The full implementation of the ATM system.

unit_test : The JUnit5 script with complete coverage tests.

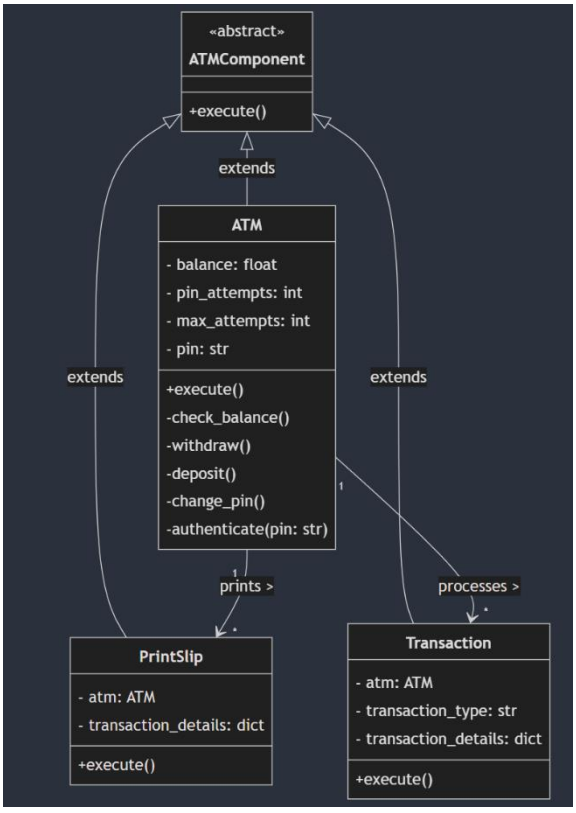
ผลลัพธ์ของ Python // Composite Design Pattern

โดย Gemini 1.5 Flash

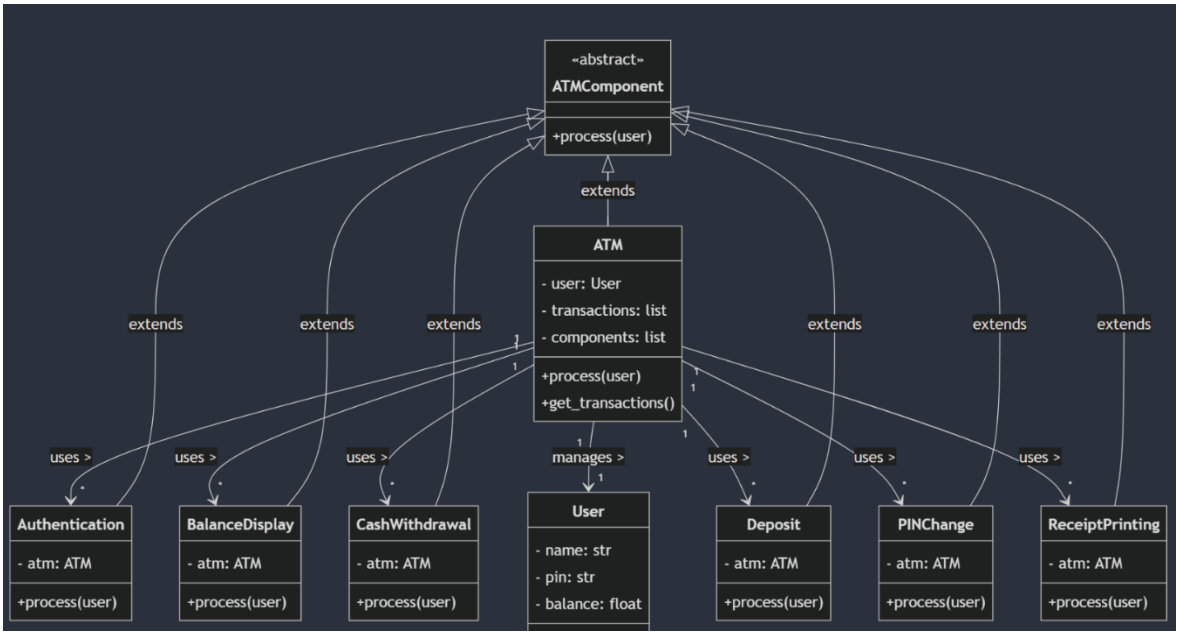
รอบที่ 1



รอบที่ 2

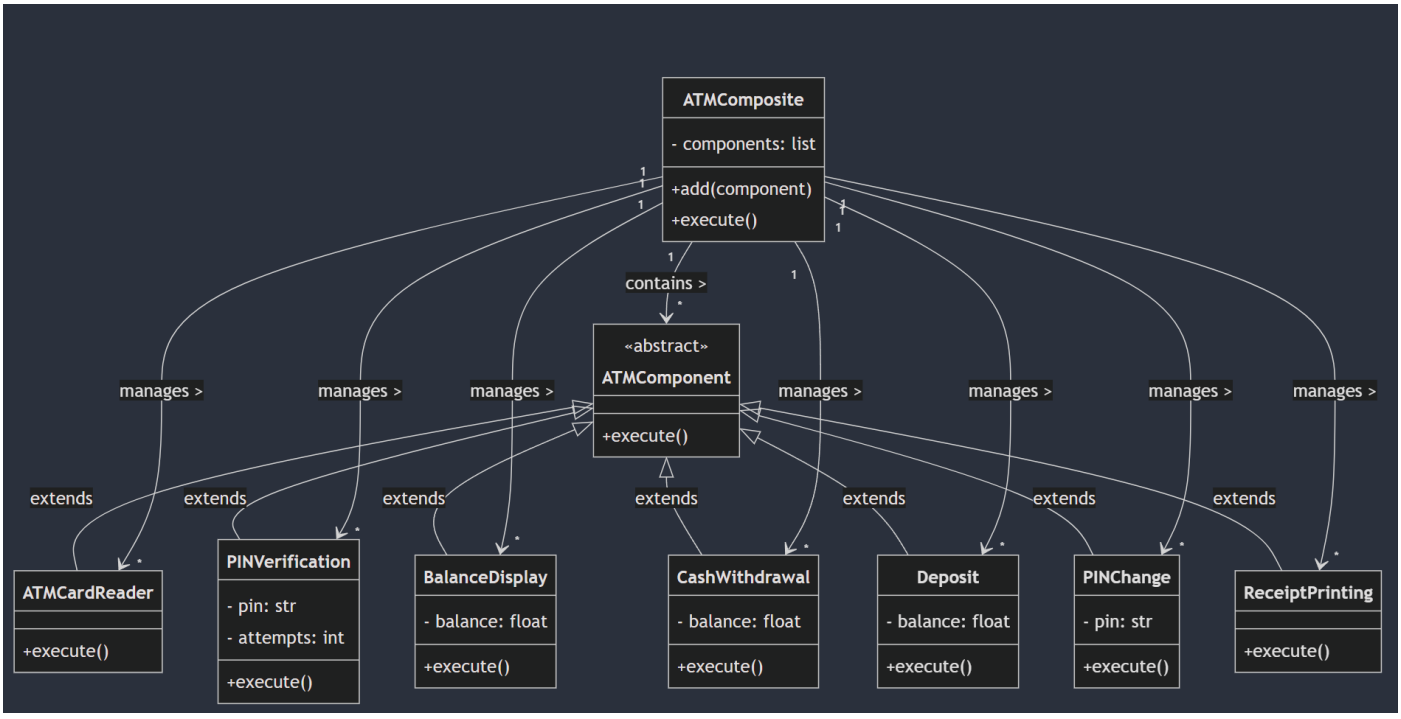


รอบที่ 3

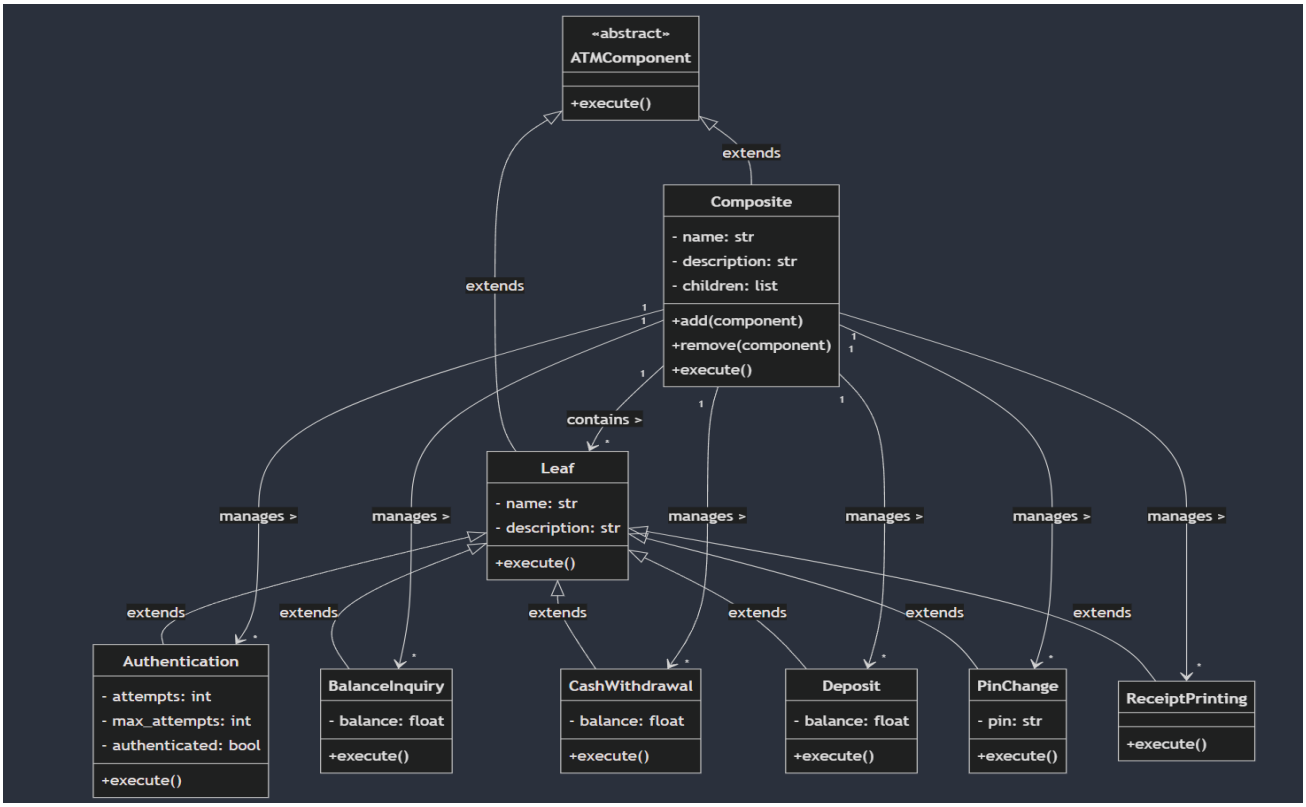


โดย Gemini 1.5 Pro

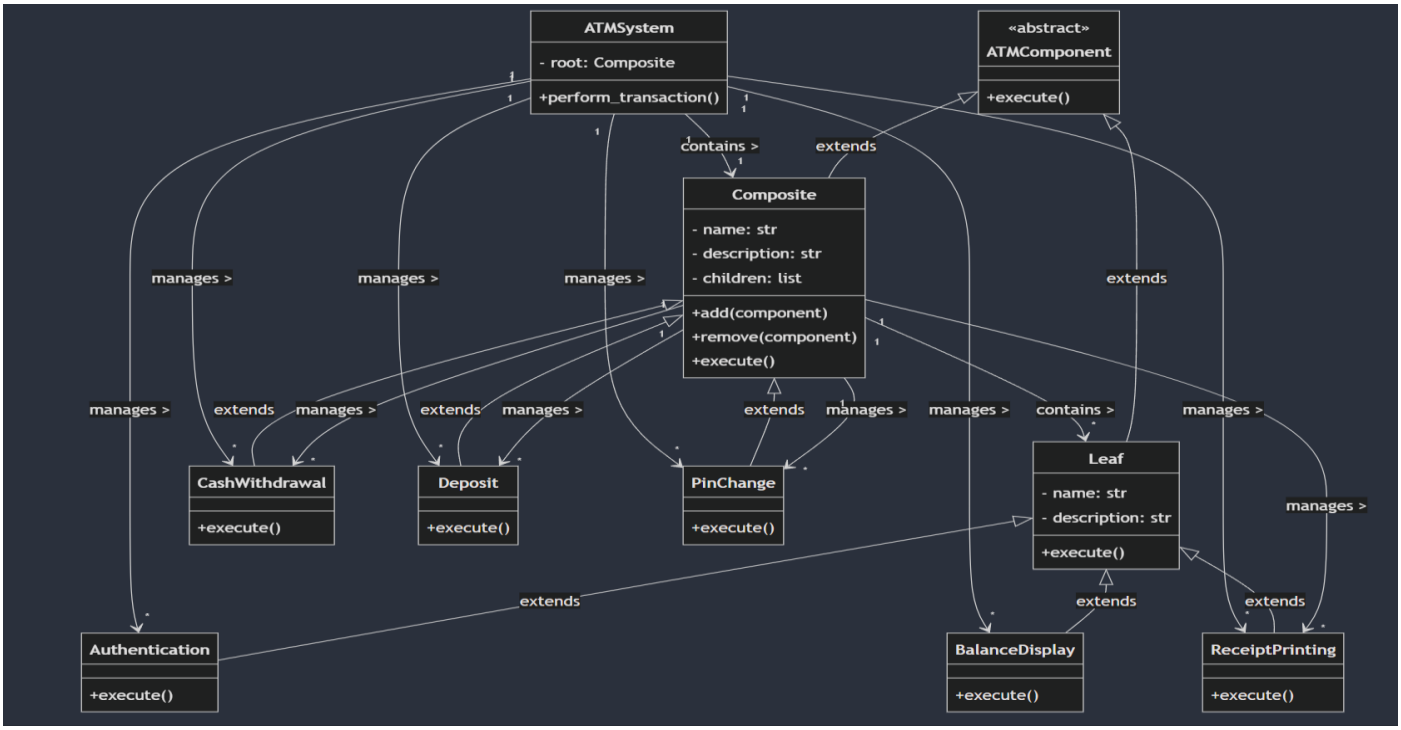
รอบที่ 1



รอบที่ 2


«abstract»
ATMComponent
+execute()
extends
Composite
- name: str
- description: str
- children: list
+add(component)
+remove(component)
+execute()
extends
contains >
Leaf
- name: str
- description: str
+execute()
manages >
manages >
manages >
manages >
manages >
manages >
extends
extends
extends
extends
extends
extends
Authentication
- attempts: int
- max_attempts: int
- authenticated: bool
+execute()
BalanceInquiry
- balance: float
+execute()
CashWithdrawal
- balance: float
+execute()
Deposit
- balance: float
+execute()
PinChange
- pin: str
+execute()
ReceiptPrinting
+execute()

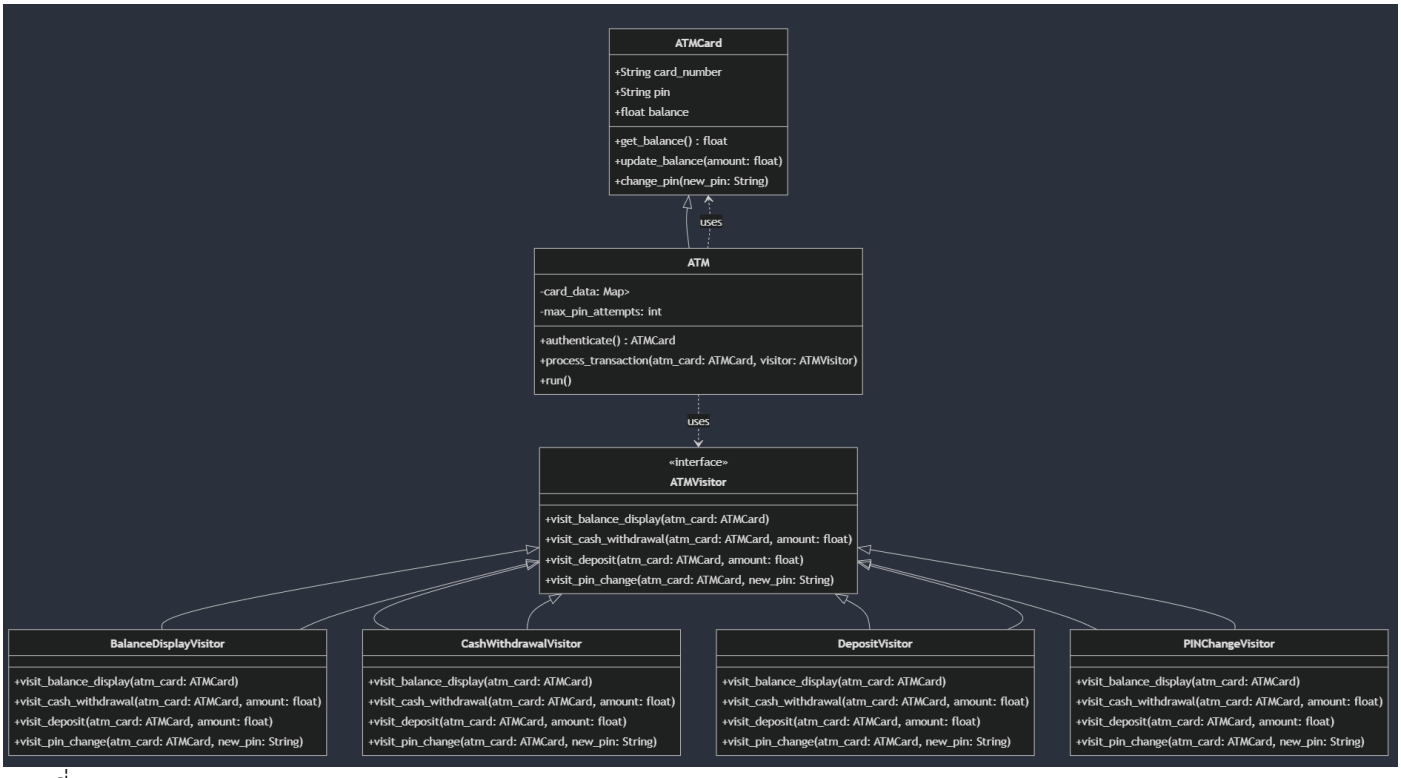
รอบที่ 3


ATMSystem
- root: Composite
+perform_transaction()
«abstract»
ATMComponent
+execute()
contains >
extends
Composite
- name: str
- description: str
- children: list
+add(component)
+remove(component)
+execute()
manages >
manages >
manages >
extends
manages >
extends
manages >
extends
manages >
extends
manages >
contains >
manages >
CashWithdrawal
+execute()
Deposit
+execute()
PinChange
+execute()
Leaf
- name: str
- description: str
+execute()
manages >
extends
extends
extends
Authentication
+execute()
BalanceDisplay
+execute()
ReceiptPrinting
+execute()

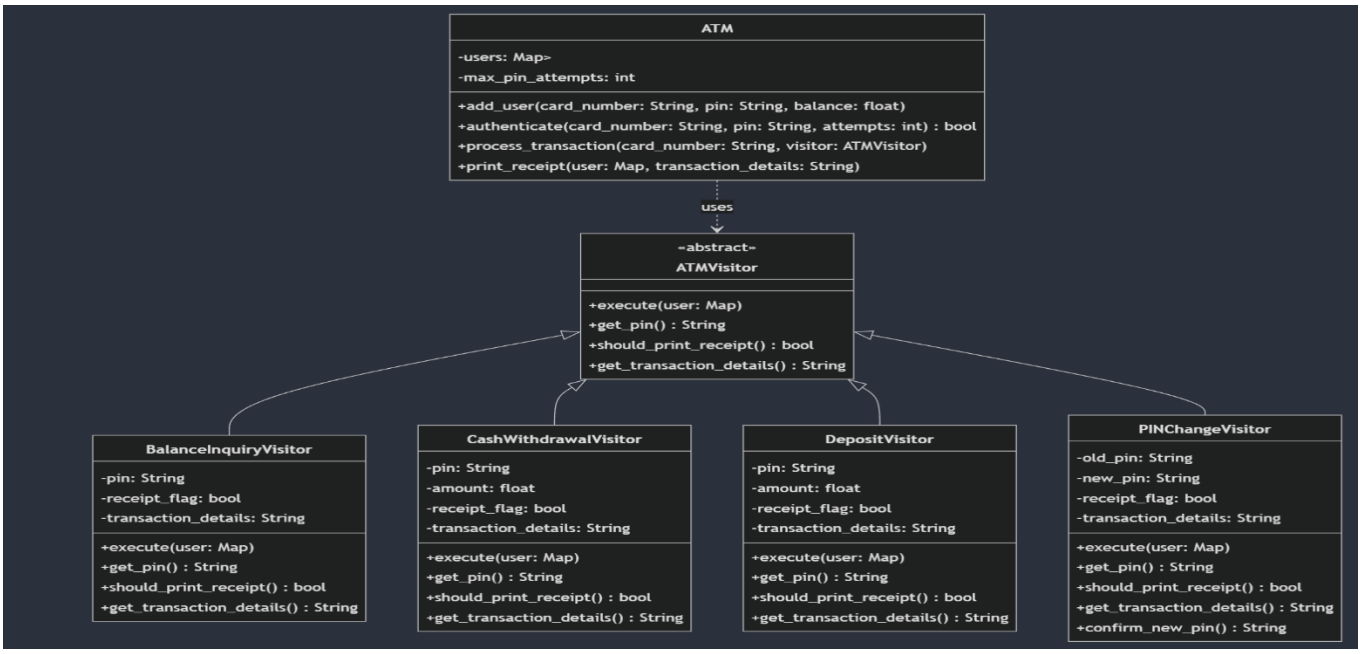
ผลลัพธ์ของ Python // Visitor Design Pattern

โดยใช้ Gemini 1.5 Flash

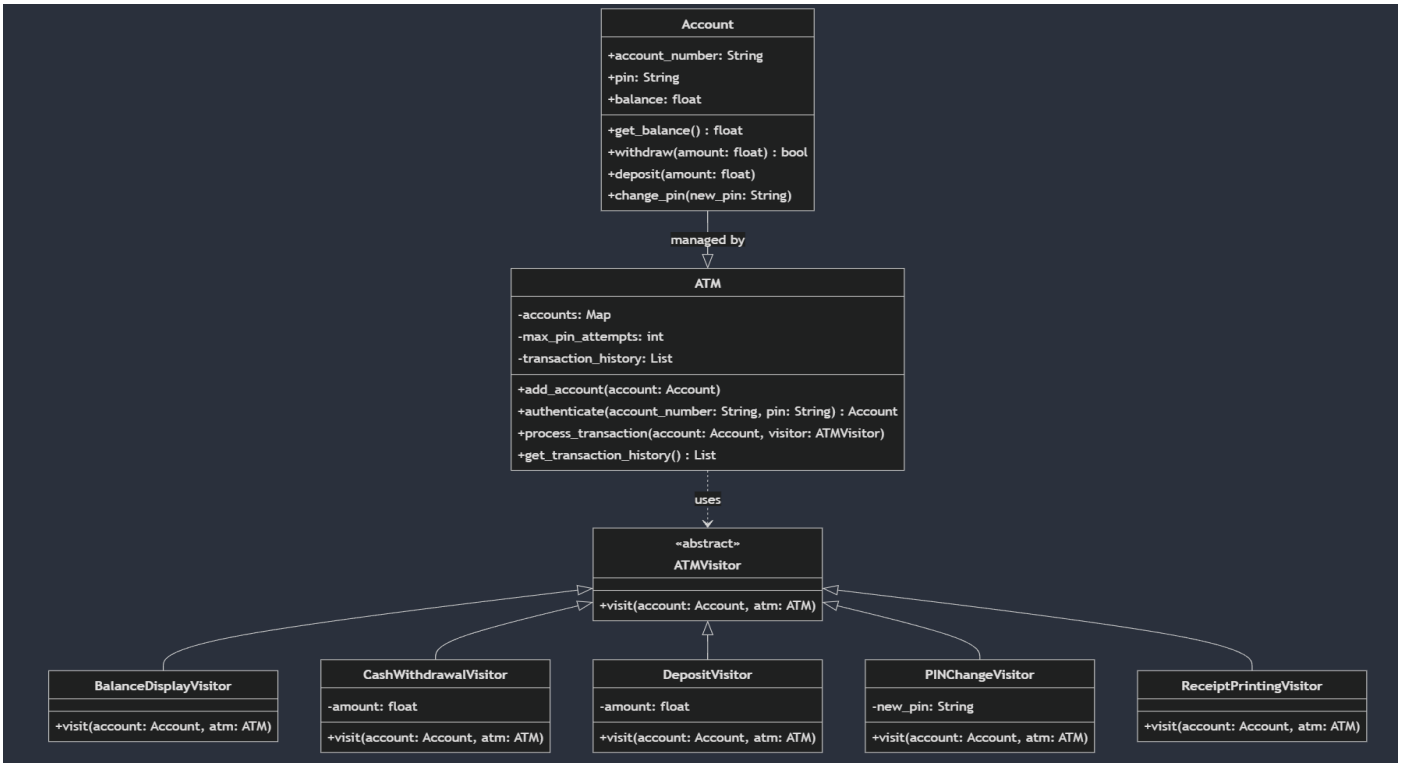
รอบที่ 1



รอบที่ 2

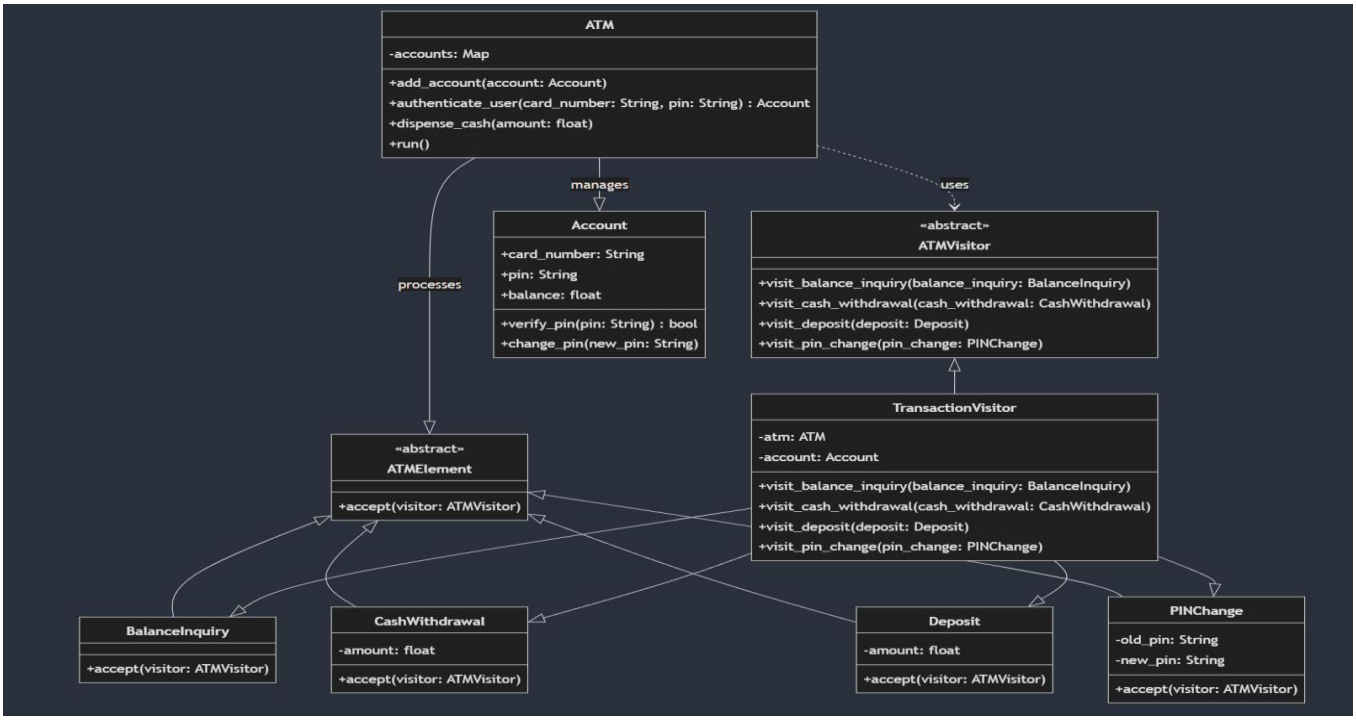


รอบที่ 3

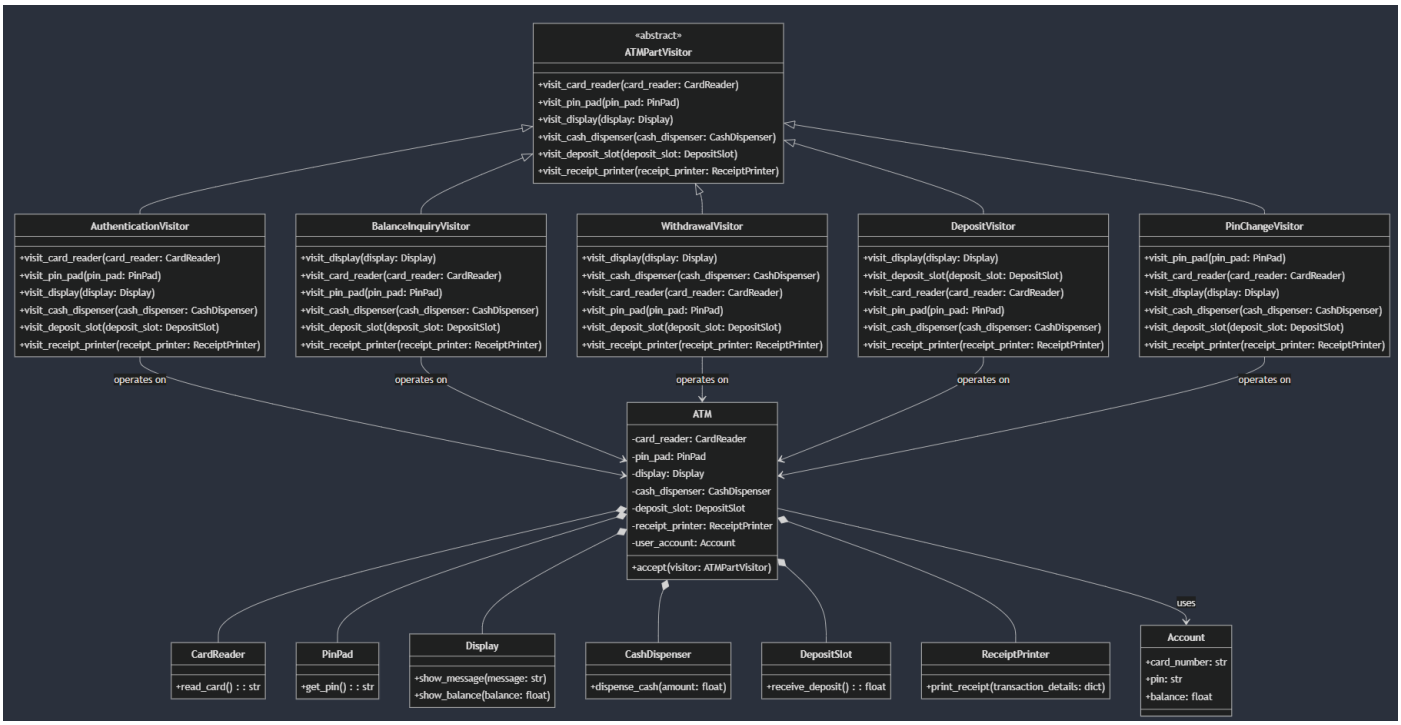

Account
+account_number: String
+pin: String
+balance: float
+get_balance() : float
+withdraw(amount: float) : bool
+deposit(amount: float)
+change_pin(new_pin: String)
managed by
ATM
-accounts: Map
-max_pin_attempts: int
-transaction_history: List
+add_account(account: Account)
+authenticate(account_number: String, pin: String) : Account
+process_transaction(account: Account, visitor: ATMVisitor)
+get_transaction_history() : List
uses
«abstract»
ATMVisitor
+visit(account: Account, atm: ATM)
BalanceDisplayVisitor
+visit(account: Account, atm: ATM)
CashWithdrawalVisitor
-amount: float
+visit(account: Account, atm: ATM)
DepositVisitor
-amount: float
+visit(account: Account, atm: ATM)
PINChangeVisitor
-new_pin: String
+visit(account: Account, atm: ATM)
ReceiptPrintingVisitor
+visit(account: Account, atm: ATM)

โดยใช้ Gemini 1.5 Pro

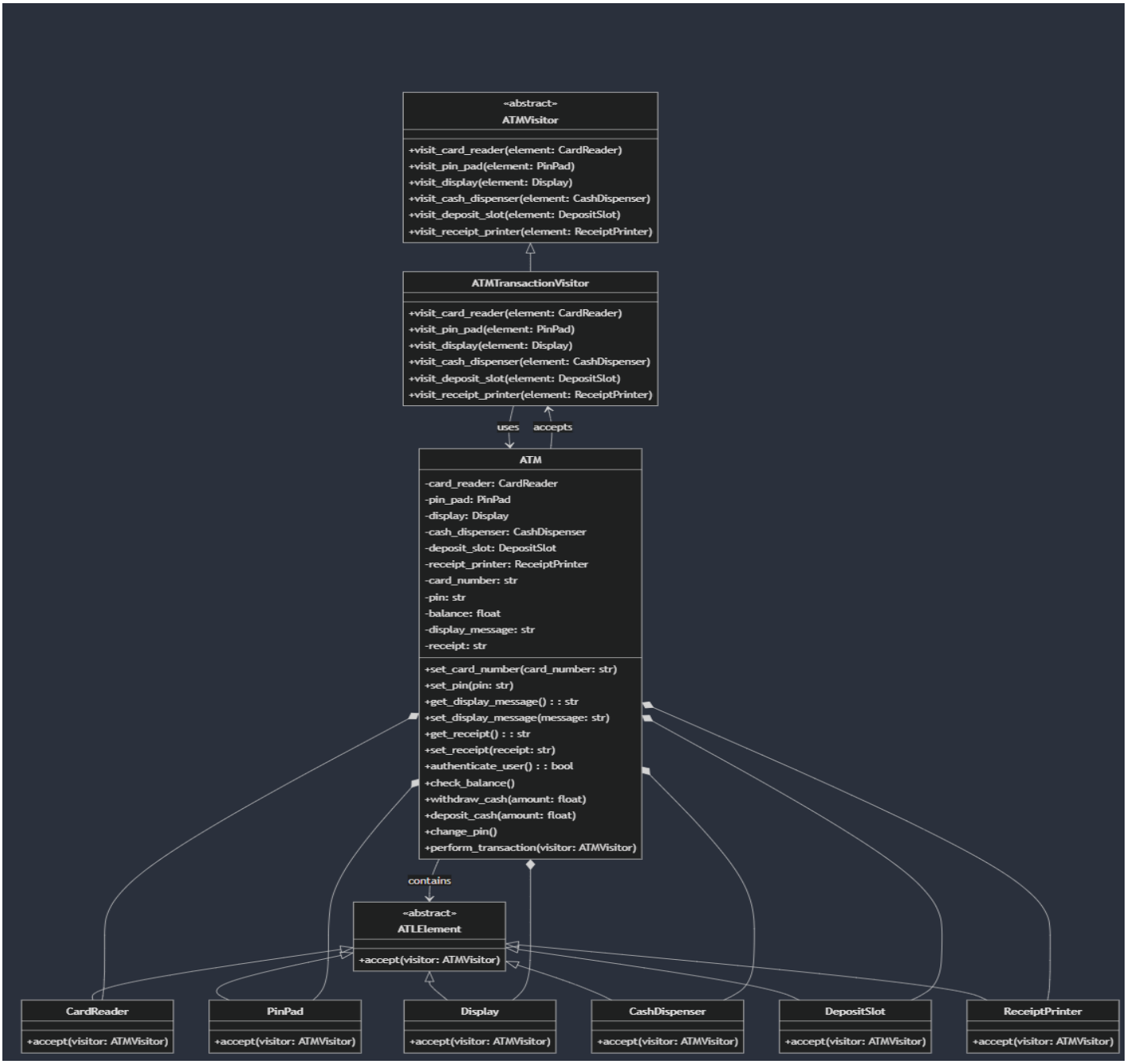
รอบที่ 1



รอบที่ 2



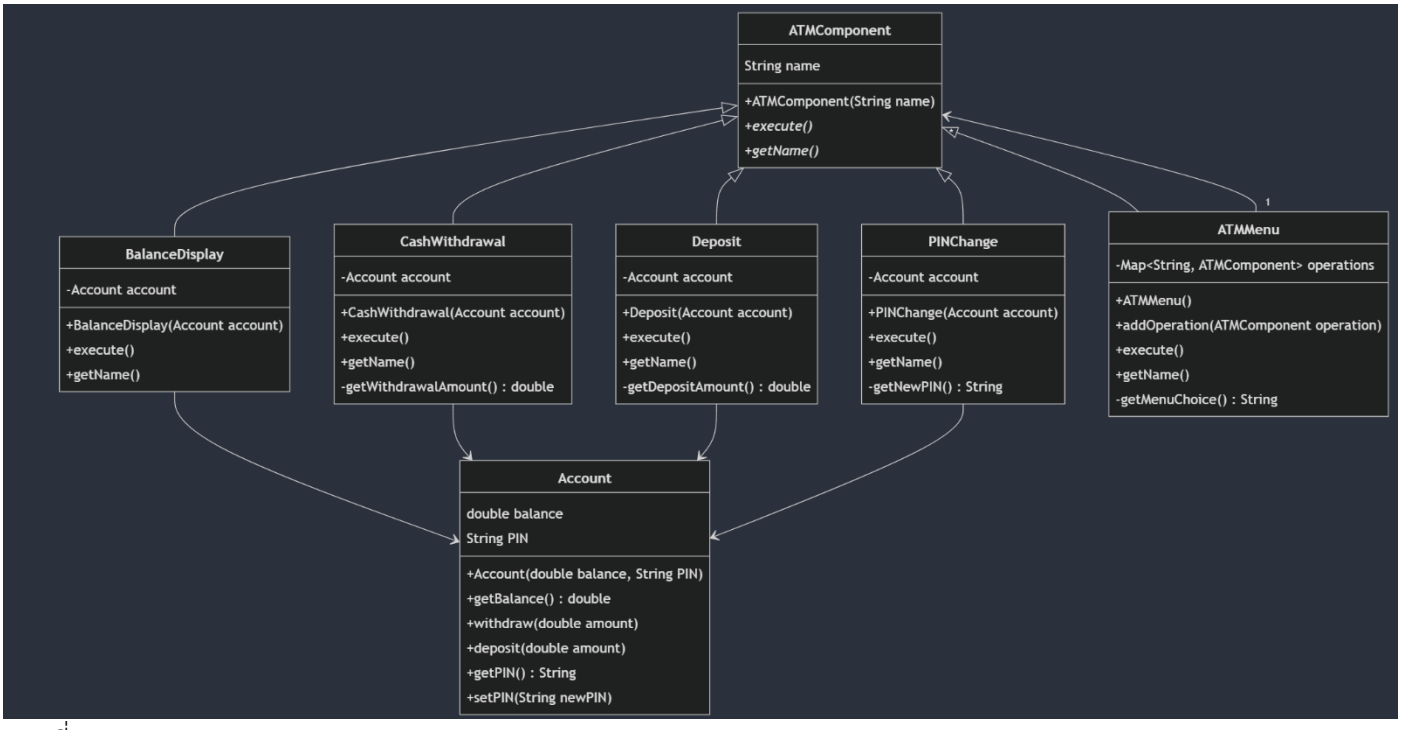
รอบที่ 3


«abstract»
ATMVisitor
+visit_card_reader(element: CardReader)
+visit_pin_pad(element: PinPad)
+visit_display(element: Display)
+visit_cash_dispenser(element: CashDispenser)
+visit_deposit_slot(element: DepositSlot)
+visit_receipt_printer(element: ReceiptPrinter)
ATMTransactionVisitor
+visit_card_reader(element: CardReader)
+visit_pin_pad(element: PinPad)
+visit_display(element: Display)
+visit_cash_dispenser(element: CashDispenser)
+visit_deposit_slot(element: DepositSlot)
+visit_receipt_printer(element: ReceiptPrinter)
uses
accepts
ATM
-card_reader: CardReader
-pin_pad: PinPad
-display: Display
-cash_dispenser: CashDispenser
-deposit_slot: DepositSlot
-receipt_printer: ReceiptPrinter
-card_number: str
-pin: str
-balance: float
-display_message: str
-receipt: str
+set_card_number(card_number: str)
+set_pin(pin: str)
+get_display_message() : : str
+set_display_message(message: str)
+get_receipt() : : str
+set_receipt(receipt: str)
+authenticate_user() : : bool
+check_balance()
+withdraw_cash(amount: float)
+deposit_cash(amount: float)
+change_pin()
+perform_transaction(visitor: ATMVisitor)
contains
«abstract»
ATLElement
+accept(visitor: ATMVisitor)
CardReader
+accept(visitor: ATMVisitor)
PinPad
+accept(visitor: ATMVisitor)
Display
+accept(visitor: ATMVisitor)
CashDispenser
+accept(visitor: ATMVisitor)
DepositSlot
+accept(visitor: ATMVisitor)
ReceiptPrinter
+accept(visitor: ATMVisitor)

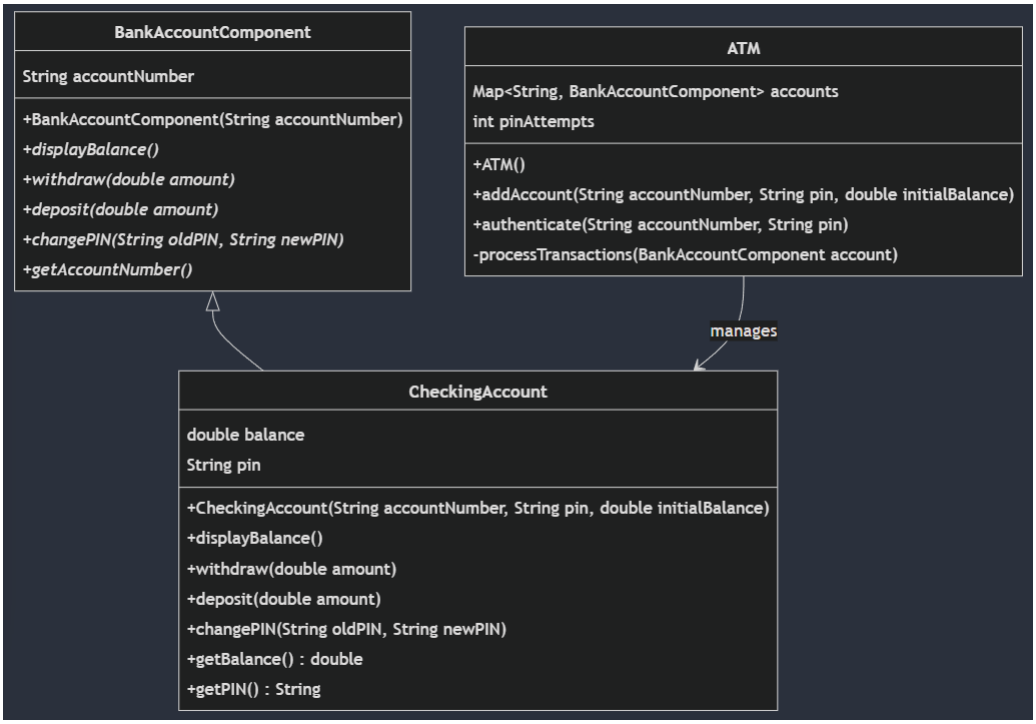
ผลลัพธ์ของ Java // Composite Design Pattern

โดยใช้ Gemini 1.5 Flash

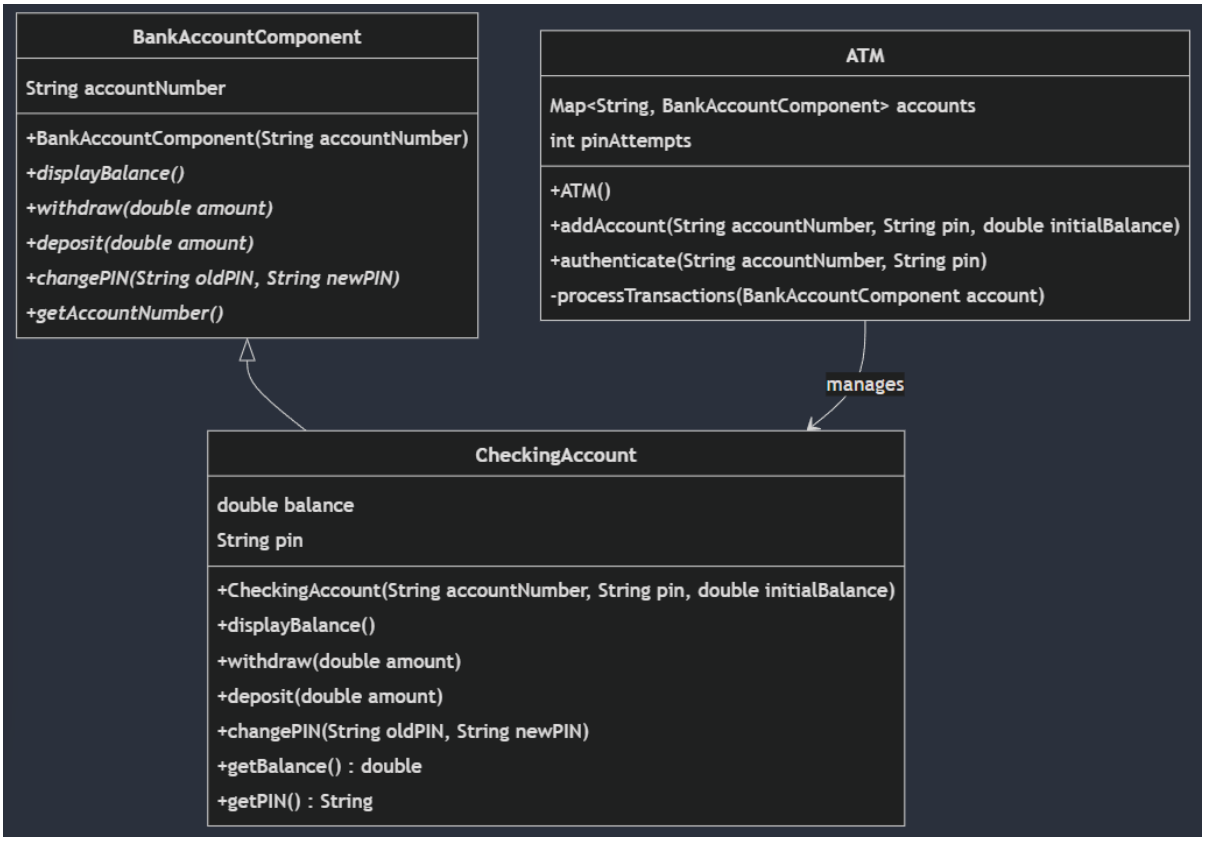
รอบที่ 1



รอบที่ 2

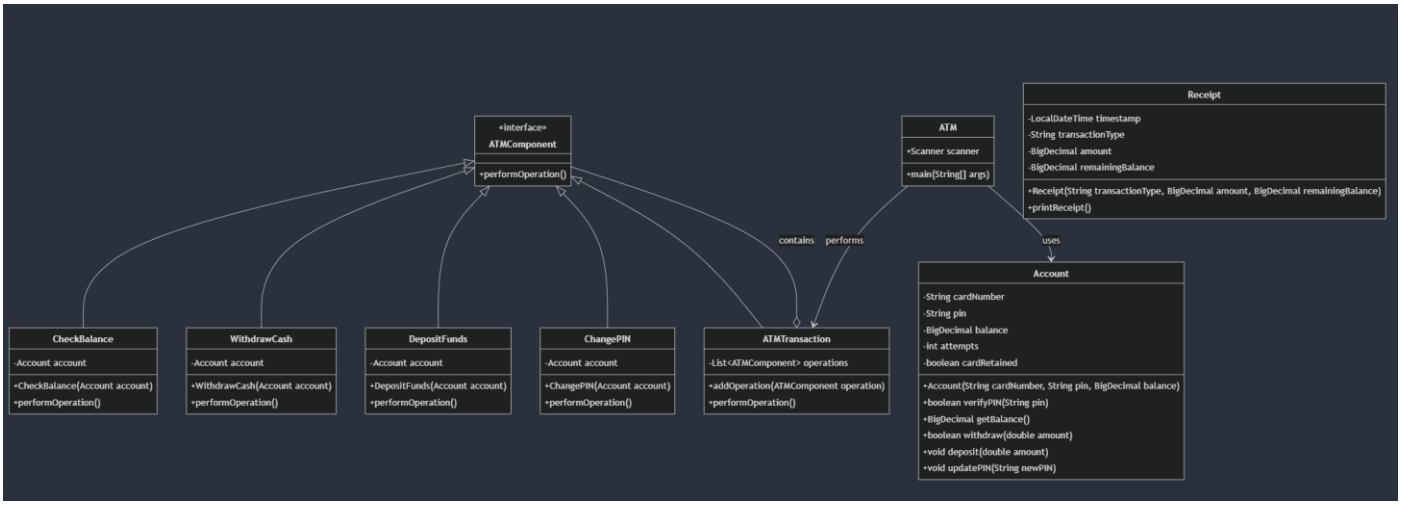

BankAccountComponent
String accountNumber
+BankAccountComponent(String accountNumber)
+displayBalance()
+withdraw(double amount)
+deposit(double amount)
+changePIN(String oldPIN, String newPIN)
+getAccountNumber()
ATM
Map<String, BankAccountComponent> accounts
int pinAttempts
+ATM()
+addAccount(String accountNumber, String pin, double initialBalance)
+authenticate(String accountNumber, String pin)
-processTransactions(BankAccountComponent account)
manages
CheckingAccount
double balance
String pin
+CheckingAccount(String accountNumber, String pin, double initialBalance)
+displayBalance()
+withdraw(double amount)
+deposit(double amount)
+changePIN(String oldPIN, String newPIN)
+getBalance() : double
+getPIN() : String

รอบที่ 3

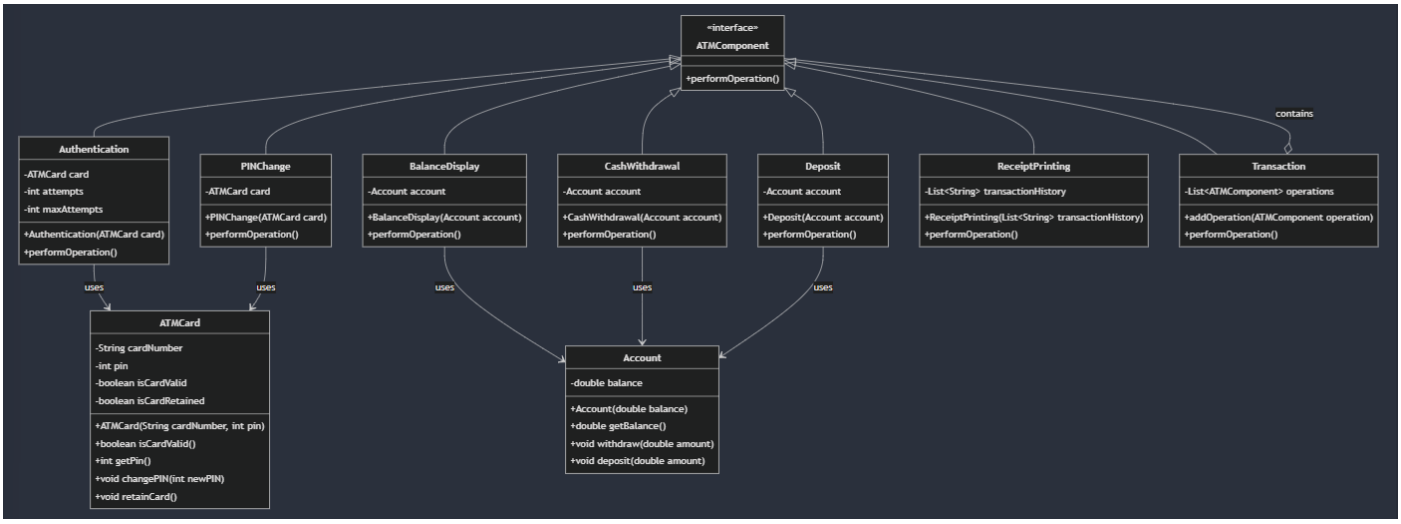

BankAccountComponent
String accountNumber
+BankAccountComponent(String accountNumber)
+displayBalance()
+withdraw(double amount)
+deposit(double amount)
+changePIN(String oldPIN, String newPIN)
+getAccountNumber()
ATM
Map<String, BankAccountComponent> accounts
int pinAttempts
+ATM()
+addAccount(String accountNumber, String pin, double initialBalance)
+authenticate(String accountNumber, String pin)
-processTransactions(BankAccountComponent account)
manages
CheckingAccount
double balance
String pin
+CheckingAccount(String accountNumber, String pin, double initialBalance)
+displayBalance()
+withdraw(double amount)
+deposit(double amount)
+changePIN(String oldPIN, String newPIN)
+getBalance() : double
+getPIN() : String

โดยใช้ Gemini 1.5 Pro

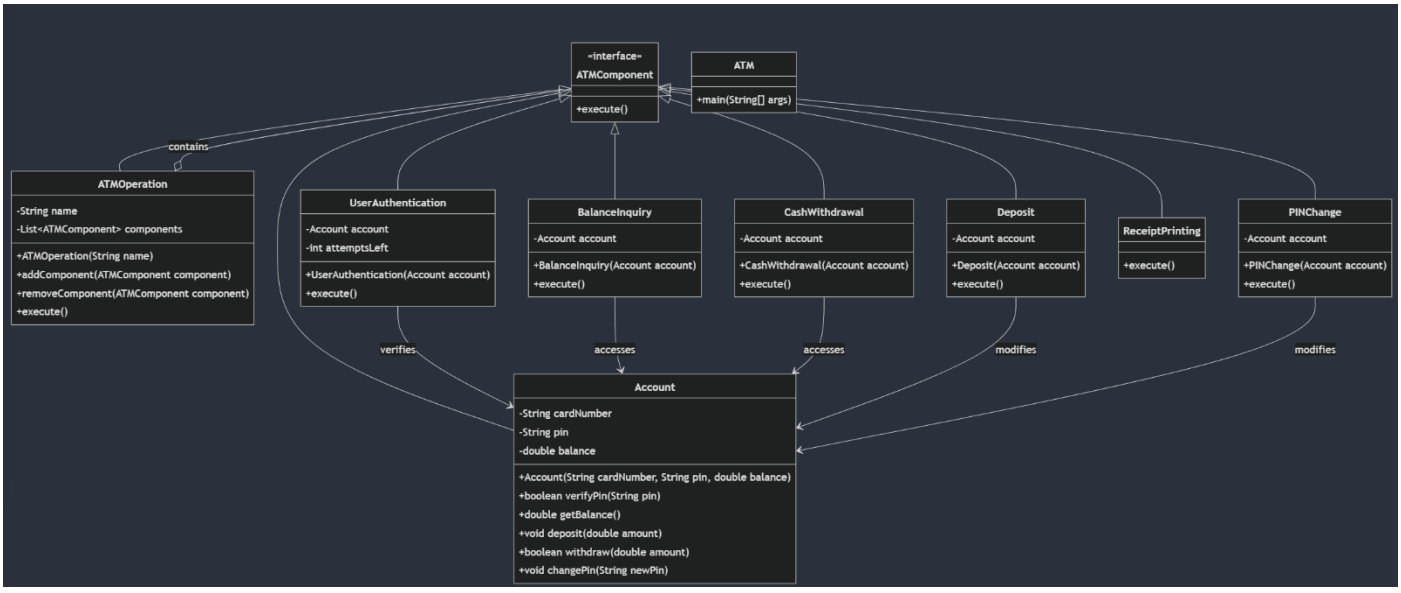
รอบที่ 1



รอบที่ 2


«interface»
ATMComponent
+performOperation()
contains
Authentication
-ATMCard card
-int attempts
-int maxAttempts
+Authentication(ATMCard card)
+performOperation()
PINChange
-ATMCard card
+PINChange(ATMCard card)
+performOperation()
BalanceDisplay
-Account account
+BalanceDisplay(Account account)
+performOperation()
CashWithdrawal
-Account account
+CashWithdrawal(Account account)
+performOperation()
Deposit
-Account account
+Deposit(Account account)
+performOperation()
ReceiptPrinting
-List<String> transactionHistory
+ReceiptPrinting(List<String> transactionHistory)
+performOperation()
Transaction
-List<ATMComponent> operations
+addOperation(ATMComponent operation)
+performOperation()
uses
uses
uses
uses
uses
ATMCard
-String cardNumber
-int pin
-boolean isCardValid
-boolean isCardRetained
+ATMCard(String cardNumber, int pin)
+boolean isCardValid()
+int getPin()
+void changePIN(int newPIN)
+void retainCard()
Account
-double balance
+Account(double balance)
+double getBalance()
+void withdraw(double amount)
+void deposit(double amount)

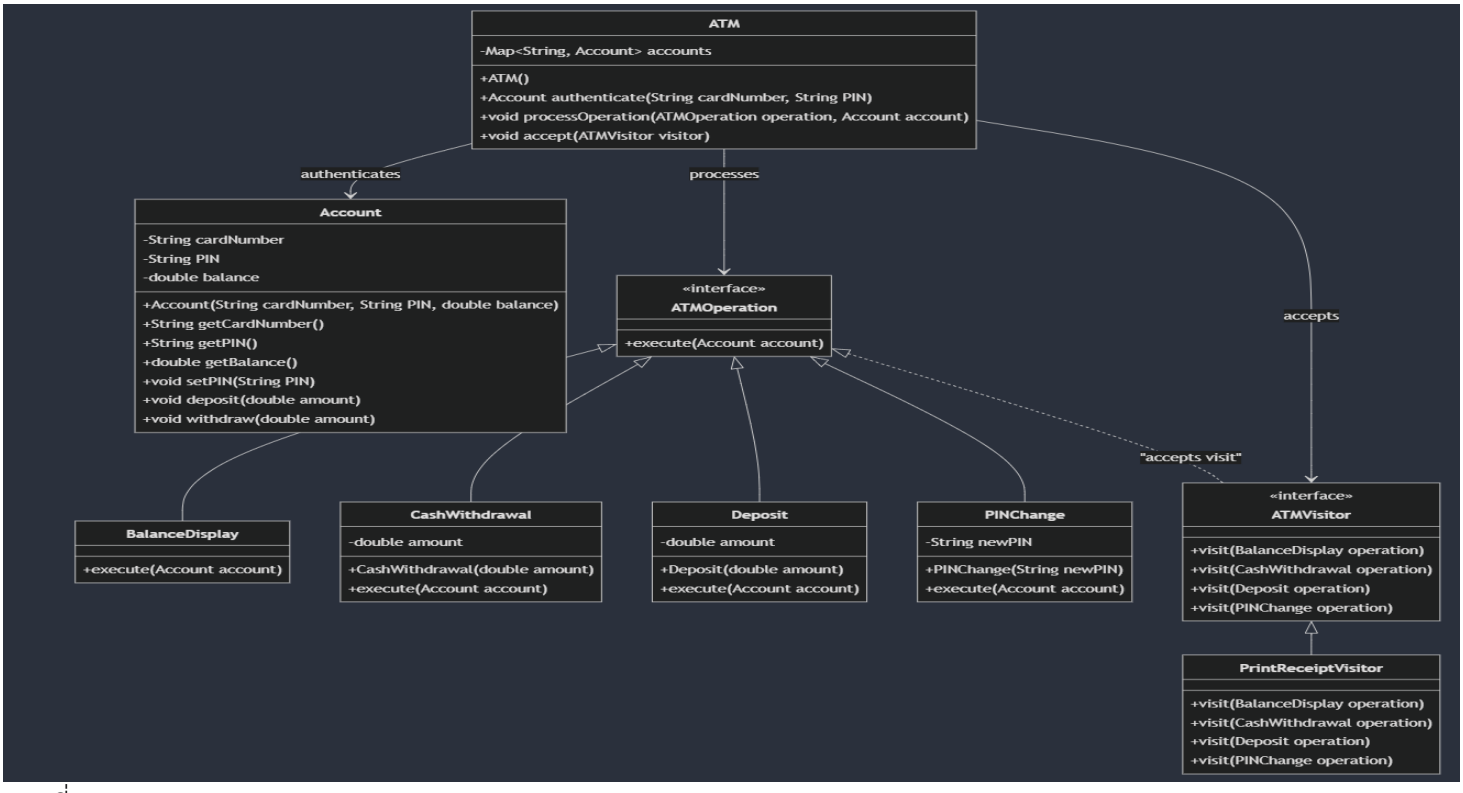
รอบที่ 3


«interface»
ATMComponent
+execute()
ATM
+main(String[] args)
contains
ATMOperation
-String name
-List<ATMComponent> components
+ATMOperation(String name)
+addComponent(ATMComponent component)
+removeComponent(ATMComponent component)
+execute()
UserAuthentication
-Account account
-int attemptsLeft
+UserAuthentication(Account account)
+execute()
BalanceInquiry
-Account account
+BalanceInquiry(Account account)
+execute()
CashWithdrawal
-Account account
+CashWithdrawal(Account account)
+execute()
Deposit
-Account account
+Deposit(Account account)
+execute()
ReceiptPrinting
+execute()
PINChange
-Account account
+PINChange(Account account)
+execute()
verifies
accesses
accesses
modifies
modifies
Account
-String cardNumber
-String pin
-double balance
+Account(String cardNumber, String pin, double balance)
+boolean verifyPin(String pin)
+double getBalance()
+void deposit(double amount)
+boolean withdraw(double amount)
+void changePin(String newPin)

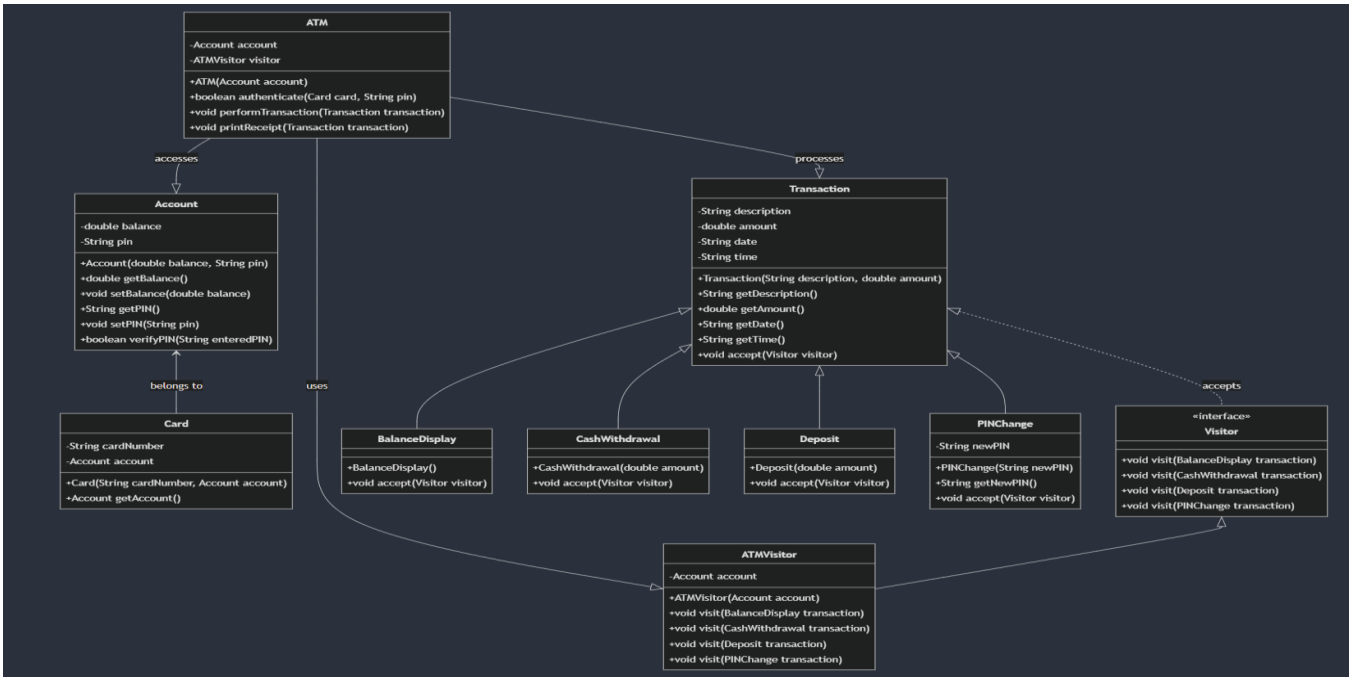
ผลลัพธ์ของ Java // Visitor Design Pattern

โดยใช้ Gemini 1.5 Flash

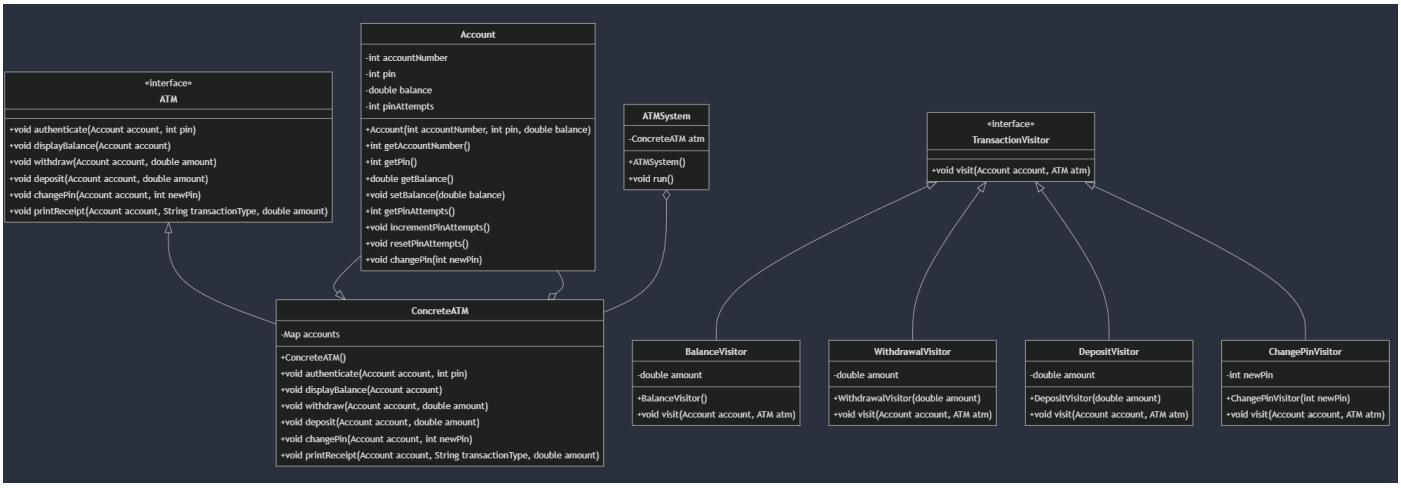
รอบที่ 1



รอบที่ 2

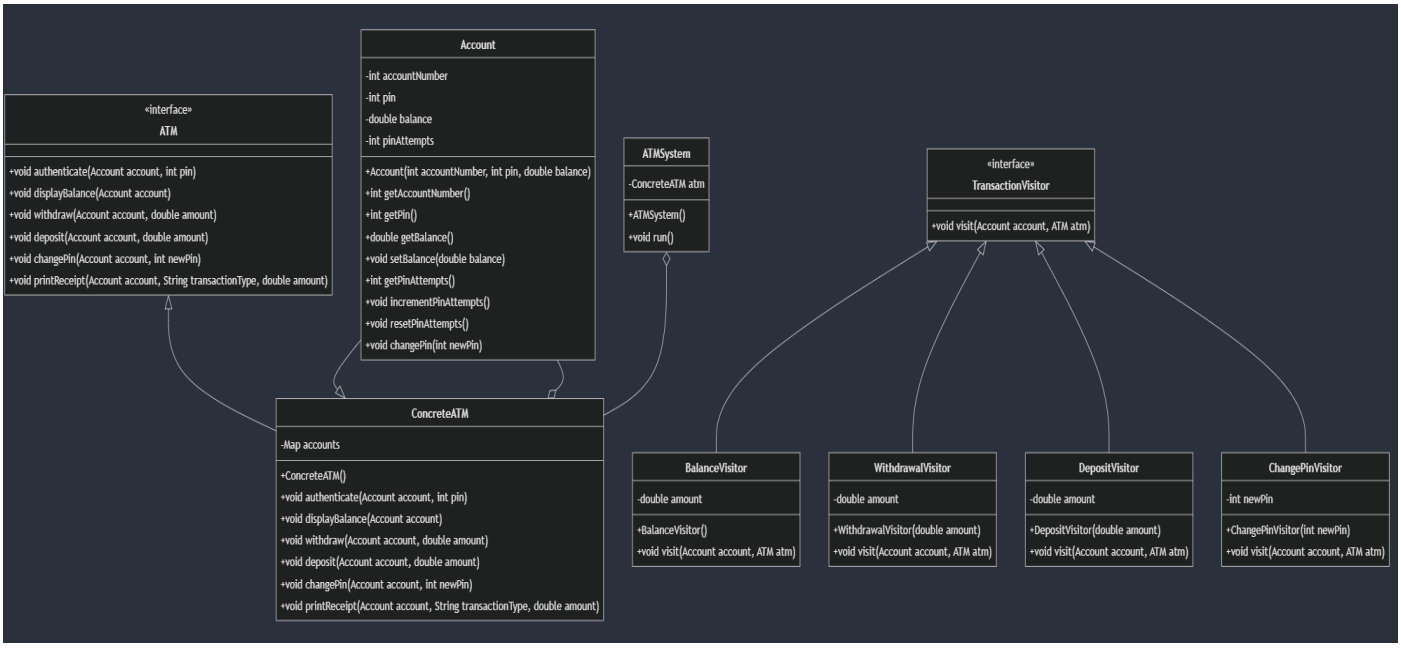


รอบที่ 3

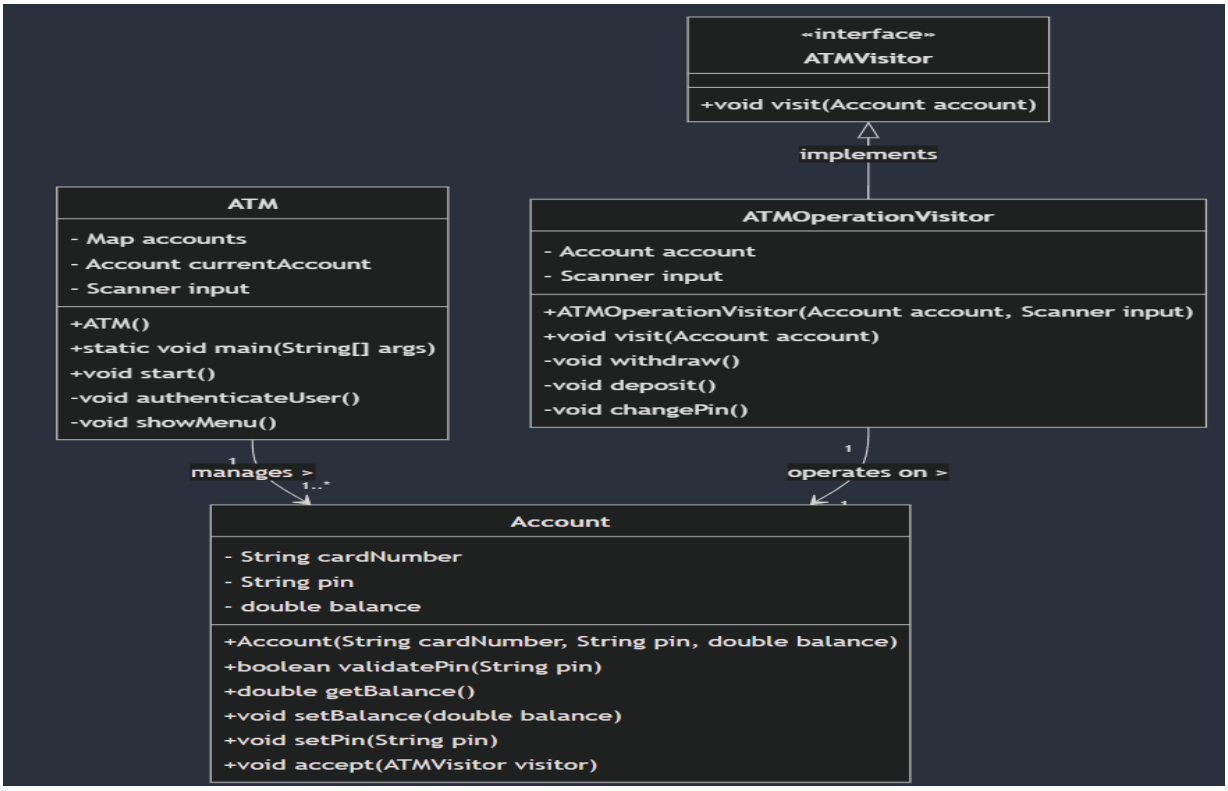


โดยใช้ Gemini 1.5 Pro

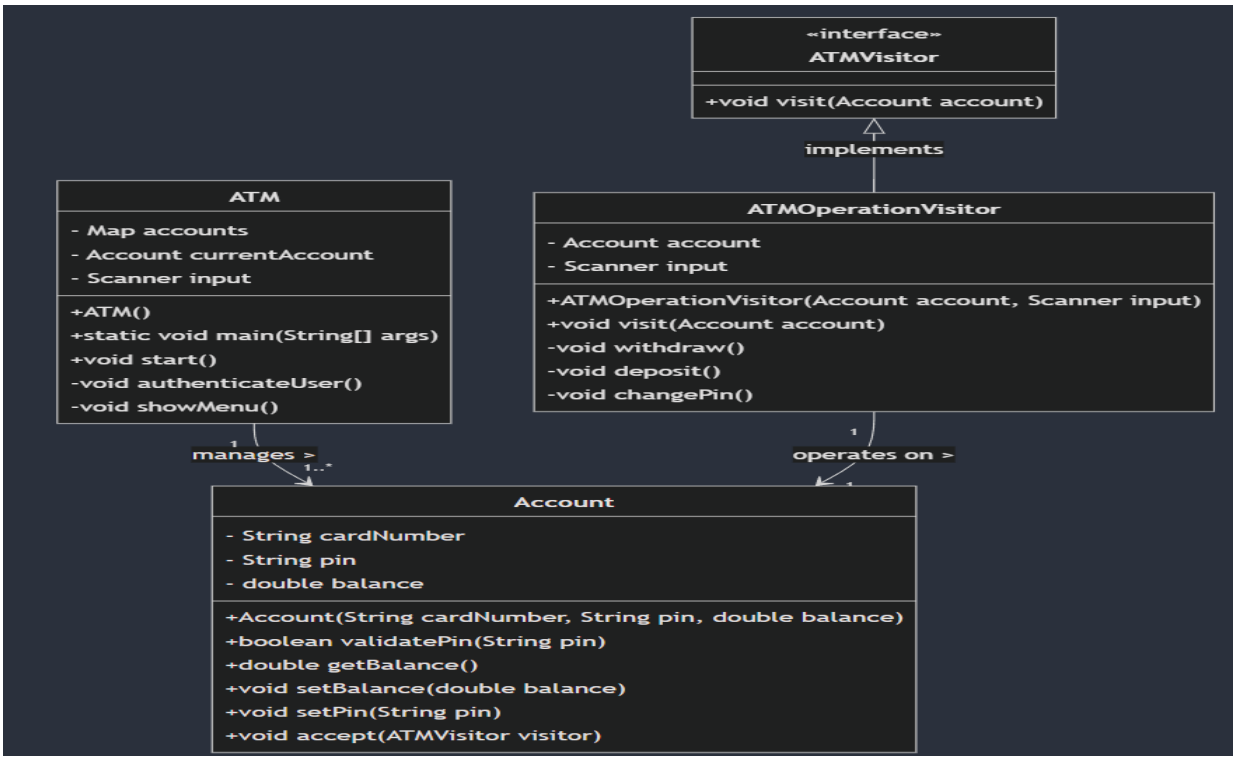
รอบที่ 1



รอบที่ 2


«interface»
ATMVisitor
+void visit(Account account)
implements
ATM
- Map accounts
- Account currentAccount
- Scanner input
+ATM()
+static void main(String[] args)
+void start()
-void authenticateUser()
-void showMenu()
ATMOperationVisitor
- Account account
- Scanner input
+ATMOperationVisitor(Account account, Scanner input)
+void visit(Account account)
-void withdraw()
-void deposit()
-void changePin()
1
manages >
1..*
1
operates on >
1
Account
- String cardNumber
- String pin
- double balance
+Account(String cardNumber, String pin, double balance)
+boolean validatePin(String pin)
+double getBalance()
+void setBalance(double balance)
+void setPin(String pin)
+void accept(ATMVisitor visitor)

รอบที่ 3


«interface»
ATMVisitor
+void visit(Account account)
implements
ATM
- Map accounts
- Account currentAccount
- Scanner input
+ATM()
+static void main(String[] args)
+void start()
-void authenticateUser()
-void showMenu()
ATMOperationVisitor
- Account account
- Scanner input
+ATMOperationVisitor(Account account, Scanner input)
+void visit(Account account)
-void withdraw()
-void deposit()
-void changePin()
1
manages >
1..*
1
operates on >
1
Account
- String cardNumber
- String pin
- double balance
+Account(String cardNumber, String pin, double balance)
+boolean validatePin(String pin)
+double getBalance()
+void setBalance(double balance)
+void setPin(String pin)
+void accept(ATMVisitor visitor)

วิเคราะห์ผลการทดลอง Gemini 1.5 Flash / Gemini 1.5 Pro

จากผลลัพธ์ พบว่า

ในส่วนของภาษาโปรแกรมมิ่งที่กำหนดให้ในพรอมพต์ไม่เป็นปัญหา โมเดลทั้งสองสามารถใช้งาน Java JUnit5 และ Python Pytest ได้อย่างถูกต้อง

ความแตกต่างหลักระหว่าง Gemini 1.5 Flash และ Gemini 1.5 Pro
ในรุ่น Flash เป็น Model ที่เน้นความไวในการสร้างคำ แต่ก็แลกมาด้วยความแม่นยำที่ต่ำลงมาทำให้โค้ดที่ได้
อาจไม่ครอบคลุมกับ Requirement ทั้งหมดที่ได้ให้ไป
ในรุ่น Pro เป็น Model ที่เน้นความแม่นยำในการสร้าง แต่จะใช้เวลานานมากกว่าตัว Flash และกินทรัพยากรมากกว่าเช่นกัน
แต่ผลลัพธ์ที่ได้ก็สามารถนำไปประยุกต์ใช้ได้ดีมากยิ่งขึ้น

Gemini 1.5 Flash สามารถสร้างโค้ดถูกต้องตาม Requirement แต่รายละเอียดการทำงานของฟังก์ชันนั้นจะไม่ซับซ้อน แล และ ในบางครั้งจะมี Design Pattern ที่ไม่ตรงกับ Requirement เช่น State Design Pattern และบางกรณีที่ขาด
ฟังก์ชั่นหลักที่ได้ให้ Requirement ไป

Gemini 1.5 Pro สามารถสร้างโครงสร้างพื้นฐานของระบบ ATM ได้ตามหลัก Composite และ Visitor Design Pattern และครอบคลุมฟังก์ชันหลักที่กำหนด แต่ยังขาดรายละเอียดบางส่วน เช่น Logic ของฟังก์ชั่นบางส่วนที่ให้มาเป็น placeholder

ในส่วนโค้ดที่ให้มีจุดผิดพลาดน้อยถึงน้อยมากในการออกแบบตาม Design Patern ที่กำหนดไว้
คาดเดาว่าปรับปรุงเพียงเล็กน้อยก็สามารถนำไปใช้งานได้

โดยสรุป โมเดล AI แบบ generative อย่าง Gemini สามารถเป็นเครื่องมือช่วยในการพัฒนาโปรแกรมได้ในระดับหนึ่ง
โดยสามารถช่วยในขั้นตอนการสร้าง ออกแบบ หรือนำร่องการพัฒนาได้ อย่างไรก็ตาม
นักพัฒนาควรมีทักษะเฉพาะทางและความเชี่ยวชาญของตนเอง

สิ่งสำคัญคือนักพัฒนาควรตรวจสอบผลลัพธ์ที่ได้จาก AI อย่างละเอียดถี่ถ้วนก่อนนำไปใช้งานจริงหรือพัฒนาต่อ
เพื่อให้มั่นใจในความถูกต้องและความสมบูรณ์ของระบบ AI สามารถให้แนวทางและช่วยเหลือได้
แต่การตัดสินใจสุดท้ายและการรับรองคุณภาพยังคงเป็นความรับผิดชอบของนักพัฒนา

การใช้ AI เป็นเครื่องมือช่วยในการพัฒนาอย่างชาญฉลาด โดยผสมผสานกับความเชี่ยวชาญของนักพัฒนา
จะช่วยเพิ่มประสิทธิภาพในกระบวนการพัฒนาซอฟต์แวร์ได้อย่างมีประสิทธิผล

3.GitHub Copilot

Prompt ที่ใช้ในการ Generate Code Python และ Generate Test ทั้งสามรอบ โดยใช้ Composite Design Pattern

Use the composite design pattern to Write code in Python programming language from the requirements below.

"ความต้องการระบบตู้ ATM

1. การยืนยันตัวตนผู้ใช้

การป้อนข้อมูล: ผู้ใช้ต้องสามารถใส่บัตร ATM และรหัส PIN ได้

การตรวจสอบ: ระบบต้องตรวจสอบความถูกต้องของบัตรและรหัส PIN

การจำกัดความพยายาม: หากใส่ผิดเกินจำนวนครั้งที่กำหนด บัตรจะถูกยึด

2. การแสดงยอดเงินคงเหลือ

การเข้าถึงข้อมูล: ผู้ใช้สามารถเลือกดูยอดเงินคงเหลือในบัญชีได้

การแสดงผล: ระบบต้องแสดงยอดเงินคงเหลือปัจจุบัน

3. การถอนเงิน

การเลือกจำนวนเงิน: ผู้ใช้สามารถเลือกจำนวนเงินที่ต้องการถอนได้

การตรวจสอบยอดเงิน: ระบบต้องตรวจสอบว่ามียอดเงินเพียงพอสำหรับการถอน

การจ่ายเงิน: หากมียอดเงินเพียงพอ ระบบจะจ่ายเงินให้ผู้ใช้ และปรับปรุงยอดเงินคงเหลือ

การแจ้งเตือน: หากมียอดเงินไม่เพียงพอ ระบบจะแจ้งเตือนผู้ใช้

4. การฝากเงิน

การรับเงินสด: ผู้ใช้สามารถใส่เงินสดเข้าเครื่อง ATM ได้

การนับเงิน: ระบบต้องนับจำนวนเงินที่ฝากเข้ามา

การปรับปรุงยอดเงิน: ระบบต้องปรับปรุงยอดเงินคงเหลือในบัญชี

5. การเปลี่ยนรหัส PIN

การเข้าถึงฟังก์ชัน: ผู้ใช้สามารถเปลี่ยนรหัส PIN ของตนเองได้

การยืนยัน: ระบบต้องขอรหัส PIN เก่าและรหัส PIN ใหม่ 2 ครั้งเพื่อยืนยัน

การเปลี่ยนรหัส: หากรหัส PIN ใหม่ตรงกัน ระบบจะเปลี่ยนรหัส PIN ให้

6. การพิมพ์สลิป

การเลือกพิมพ์: หลังจากทำรายการเสร็จสิ้น ผู้ใช้สามารถเลือกพิมพ์สลิปได้

เนื้อหาสลิป: สลิปต้องแสดงข้อมูลรายละเอียดของรายการ เช่น วันที่ เวลา จำนวนเงิน และยอดเงินคงเหลือ"

Write pytest to test the given code that have 100% statement coverage and 100% Branch coverage.

Prompt ที่ใช้ในการ Generate Code Python และ Generate Test ทั้งสามรอบ โดยใช้ Visitor Design Pattern

Use the visitor design pattern to Write code in Python programming language from the requirements below.

"ความต้องการระบบตู้ ATM

1. การยืนยันตัวตนผู้ใช้

การป้อนข้อมูล: ผู้ใช้ต้องสามารถใส่บัตร ATM และรหัส PIN ได้

การตรวจสอบ: ระบบต้องตรวจสอบความถูกต้องของบัตรและรหัส PIN

การจำกัดความพยายาม: หากใส่ผิดเกินจำนวนครั้งที่กำหนด บัตรจะถูกยึด

2. การแสดงยอดเงินคงเหลือ

การเข้าถึงข้อมูล: ผู้ใช้สามารถเลือกดูยอดเงินคงเหลือในบัญชีได้

การแสดงผล: ระบบต้องแสดงยอดเงินคงเหลือปัจจุบัน

3. การถอนเงิน

การเลือกจำนวนเงิน: ผู้ใช้สามารถเลือกจำนวนเงินที่ต้องการถอนได้

การตรวจสอบยอดเงิน: ระบบต้องตรวจสอบว่ามียอดเงินเพียงพอสำหรับการถอน

การจ่ายเงิน: หากมียอดเงินเพียงพอ ระบบจะจ่ายเงินให้ผู้ใช้ และปรับปรุงยอดเงินคงเหลือ

การแจ้งเตือน: หากมียอดเงินไม่เพียงพอ ระบบจะแจ้งเตือนผู้ใช้

4. การฝากเงิน

การรับเงินสด: ผู้ใช้สามารถใส่เงินสดเข้าเครื่อง ATM ได้

การนับเงิน: ระบบต้องนับจำนวนเงินที่ฝากเข้ามา

การปรับปรุงยอดเงิน: ระบบต้องปรับปรุงยอดเงินคงเหลือในบัญชี

5. การเปลี่ยนรหัส PIN

การเข้าถึงฟังก์ชัน: ผู้ใช้สามารถเปลี่ยนรหัส PIN ของตนเองได้

การยืนยัน: ระบบต้องขอรหัส PIN เก่าและรหัส PIN ใหม่ 2 ครั้งเพื่อยืนยัน

การเปลี่ยนรหัส: หากรหัส PIN ใหม่ตรงกัน ระบบจะเปลี่ยนรหัส PIN ให้

6. การพิมพ์สลิป

การเลือกพิมพ์: หลังจากทำรายการเสร็จสิ้น ผู้ใช้สามารถเลือกพิมพ์สลิปได้

เนื้อหาสลิป: สลิปต้องแสดงข้อมูลรายละเอียดของรายการ เช่น วันที่ เวลา จำนวนเงิน และยอดเงินคงเหลือ"

Write pytest to test the given code that have 100% statement coverage and 100% Branch coverage.

Prompt ที่ใช้ในการ Generate Code Java และ Generate Test ทั้งสามรอบ โดยใช้ Composite Design Pattern

Use the composite design pattern to Write code in Java programming language from the requirements below.

"ความต้องการระบบตู้ ATM

1. การยืนยันตัวตนผู้ใช้

การป้อนข้อมูล: ผู้ใช้ต้องสามารถใส่บัตร ATM และรหัส PIN ได้

การตรวจสอบ: ระบบต้องตรวจสอบความถูกต้องของบัตรและรหัส PIN

การจำกัดความพยายาม: หากใส่ผิดเกินจำนวนครั้งที่กำหนด บัตรจะถูกยึด

2. การแสดงยอดเงินคงเหลือ

การเข้าถึงข้อมูล: ผู้ใช้สามารถเลือกดูยอดเงินคงเหลือในบัญชีได้

การแสดงผล: ระบบต้องแสดงยอดเงินคงเหลือปัจจุบัน

3. การถอนเงิน

การเลือกจำนวนเงิน: ผู้ใช้สามารถเลือกจำนวนเงินที่ต้องการถอนได้

การตรวจสอบยอดเงิน: ระบบต้องตรวจสอบว่ามียอดเงินเพียงพอสำหรับการถอน

การจ่ายเงิน: หากมียอดเงินเพียงพอ ระบบจะจ่ายเงินให้ผู้ใช้ และปรับปรุงยอดเงินคงเหลือ

การแจ้งเตือน: หากมียอดเงินไม่เพียงพอ ระบบจะแจ้งเตือนผู้ใช้

4. การฝากเงิน

การรับเงินสด: ผู้ใช้สามารถใส่เงินสดเข้าเครื่อง ATM ได้

การนับเงิน: ระบบต้องนับจำนวนเงินที่ฝากเข้ามา

การปรับปรุงยอดเงิน: ระบบต้องปรับปรุงยอดเงินคงเหลือในบัญชี

5. การเปลี่ยนรหัส PIN

การเข้าถึงฟังก์ชัน: ผู้ใช้สามารถเปลี่ยนรหัส PIN ของตนเองได้

การยืนยัน: ระบบต้องขอรหัส PIN เก่าและรหัส PIN ใหม่ 2 ครั้งเพื่อยืนยัน

การเปลี่ยนรหัส: หากรหัส PIN ใหม่ตรงกัน ระบบจะเปลี่ยนรหัส PIN ให้

6. การพิมพ์สลิป

การเลือกพิมพ์: หลังจากทำรายการเสร็จสิ้น ผู้ใช้สามารถเลือกพิมพ์สลิปได้

เนื้อหาสลิป: สลิปต้องแสดงข้อมูลรายละเอียดของรายการ เช่น วันที่ เวลา จำนวนเงิน และยอดเงินคงเหลือ"

Write to test the given code that have 100% statement coverage and 100% Branch coverage.

Prompt ที่ใช้ในการ Generate Code Java และ Generate Test ทั้งสามรอบ โดยใช้ Visitor Design Pattern

Use the visitor design pattern to Write code in Java programming language from the requirements below.

"ความต้องการระบบตู้ ATM

1. การยืนยันตัวตนผู้ใช้

การป้อนข้อมูล: ผู้ใช้ต้องสามารถใส่บัตร ATM และรหัส PIN ได้

การตรวจสอบ: ระบบต้องตรวจสอบความถูกต้องของบัตรและรหัส PIN

การจำกัดความพยายาม: หากใส่ผิดเกินจำนวนครั้งที่กำหนด บัตรจะถูกยึด

2. การแสดงยอดเงินคงเหลือ

การเข้าถึงข้อมูล: ผู้ใช้สามารถเลือกดูยอดเงินคงเหลือในบัญชีได้

การแสดงผล: ระบบต้องแสดงยอดเงินคงเหลือปัจจุบัน

3. การถอนเงิน

การเลือกจำนวนเงิน: ผู้ใช้สามารถเลือกจำนวนเงินที่ต้องการถอนได้

การตรวจสอบยอดเงิน: ระบบต้องตรวจสอบว่ามียอดเงินเพียงพอสำหรับการถอน

การจ่ายเงิน: หากมียอดเงินเพียงพอ ระบบจะจ่ายเงินให้ผู้ใช้ และปรับปรุงยอดเงินคงเหลือ

การแจ้งเตือน: หากมียอดเงินไม่เพียงพอ ระบบจะแจ้งเตือนผู้ใช้

4. การฝากเงิน

การรับเงินสด: ผู้ใช้สามารถใส่เงินสดเข้าเครื่อง ATM ได้

การนับเงิน: ระบบต้องนับจำนวนเงินที่ฝากเข้ามา

การปรับปรุงยอดเงิน: ระบบต้องปรับปรุงยอดเงินคงเหลือในบัญชี

5. การเปลี่ยนรหัส PIN

การเข้าถึงฟังก์ชัน: ผู้ใช้สามารถเปลี่ยนรหัส PIN ของตนเองได้

การยืนยัน: ระบบต้องขอรหัส PIN เก่าและรหัส PIN ใหม่ 2 ครั้งเพื่อยืนยัน

การเปลี่ยนรหัส: หากรหัส PIN ใหม่ตรงกัน ระบบจะเปลี่ยนรหัส PIN ให้

6. การพิมพ์สลิป

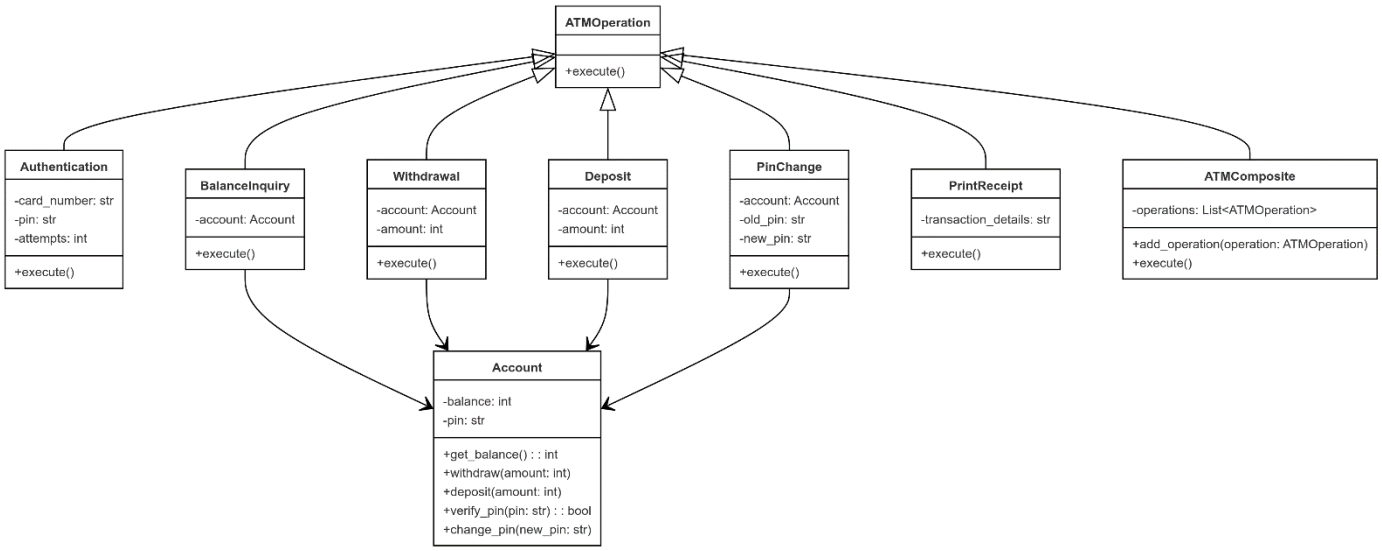
การเลือกพิมพ์: หลังจากทำรายการเสร็จสิ้น ผู้ใช้สามารถเลือกพิมพ์สลิปได้

เนื้อหาสลิป: สลิปต้องแสดงข้อมูลรายละเอียดของรายการ เช่น วันที่ เวลา จำนวนเงิน และยอดเงินคงเหลือ"

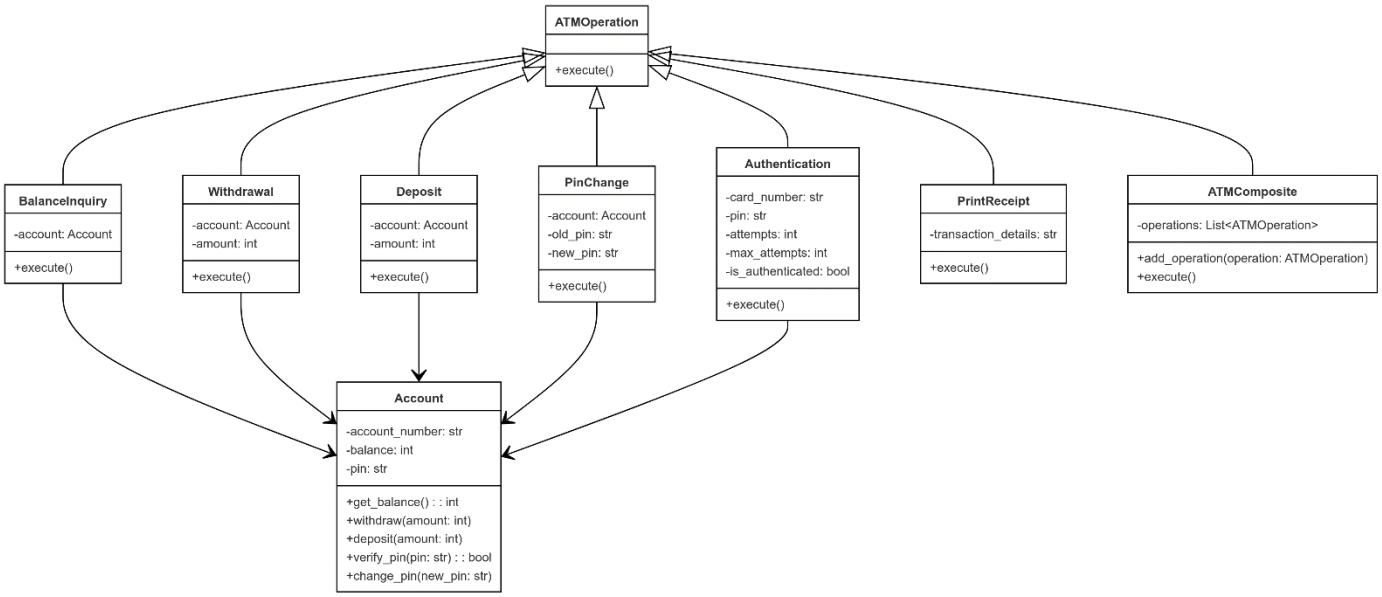
Write to test the given code that **have** 100% statement coverage and 100% Branch coverage.

ผลลัพธ์ของ Python // Composite Design Pattern

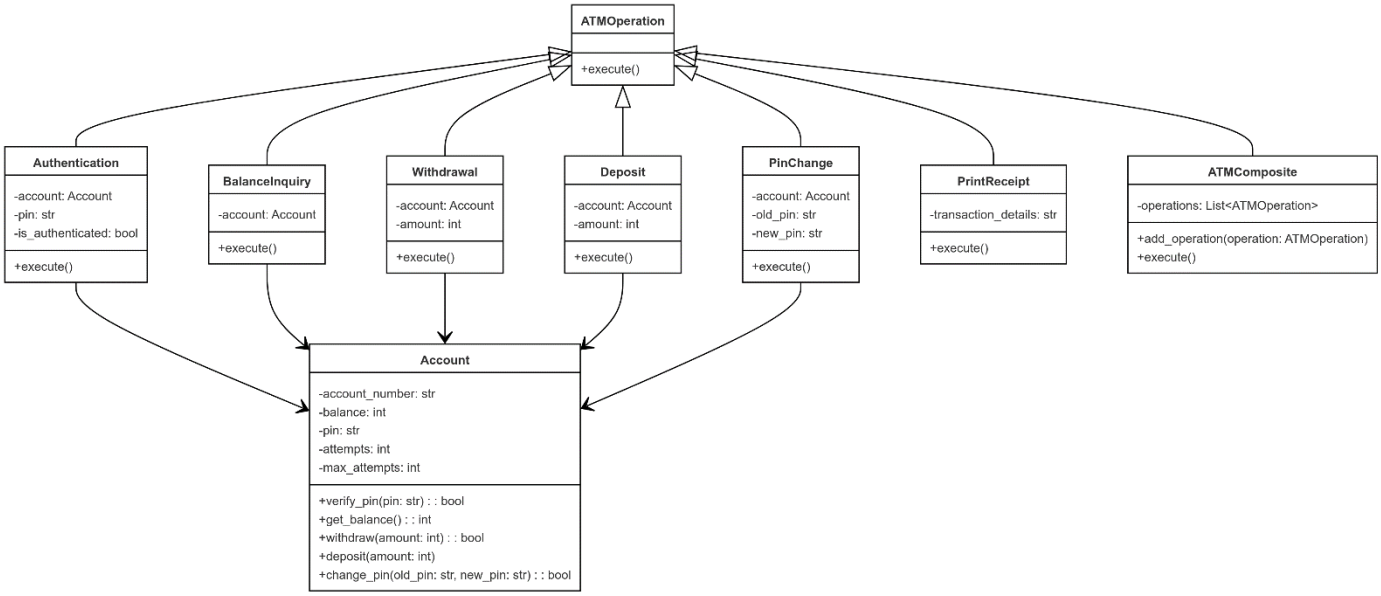
รอบที่1



รอบที่2

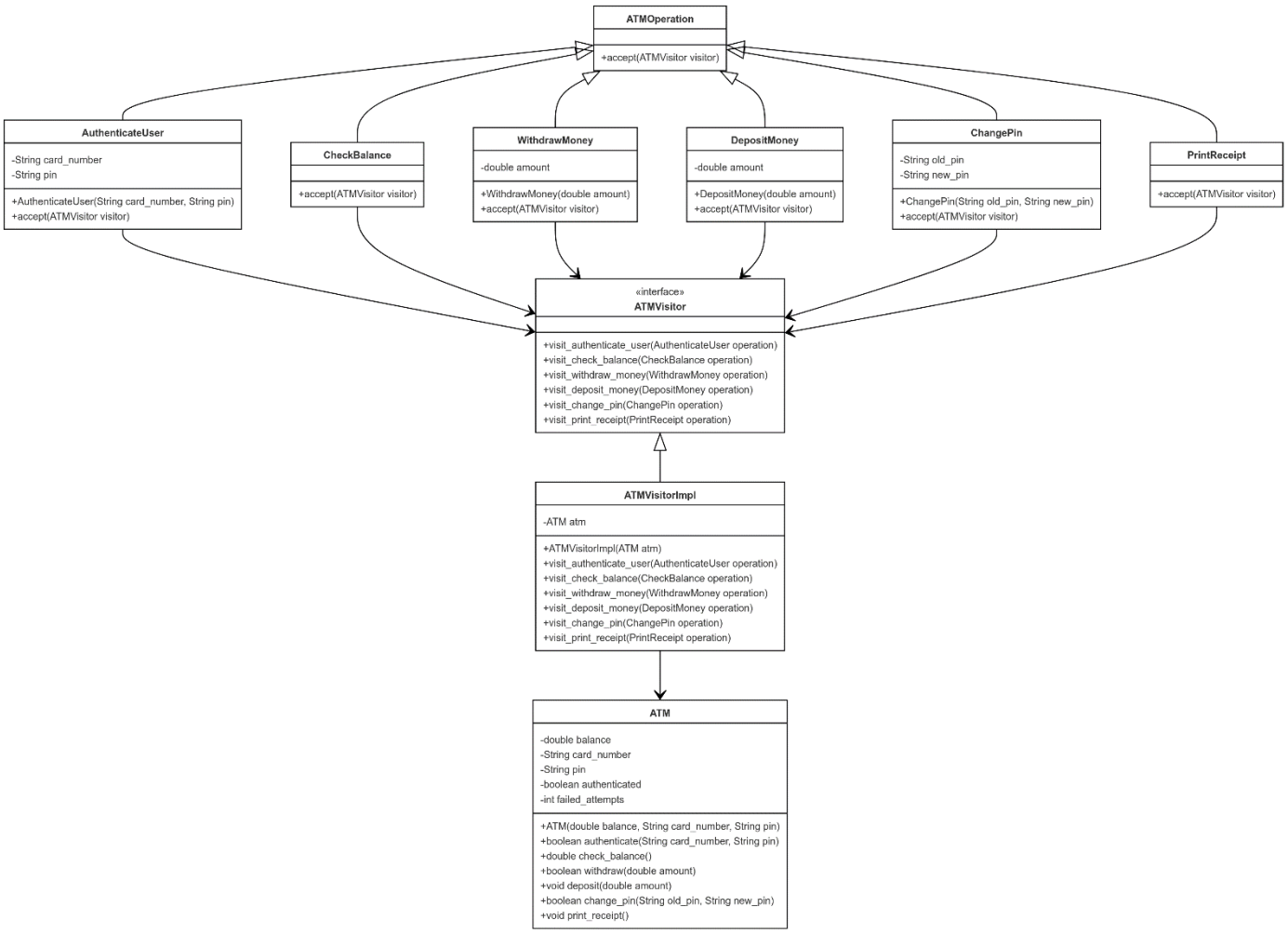


รอบที่3

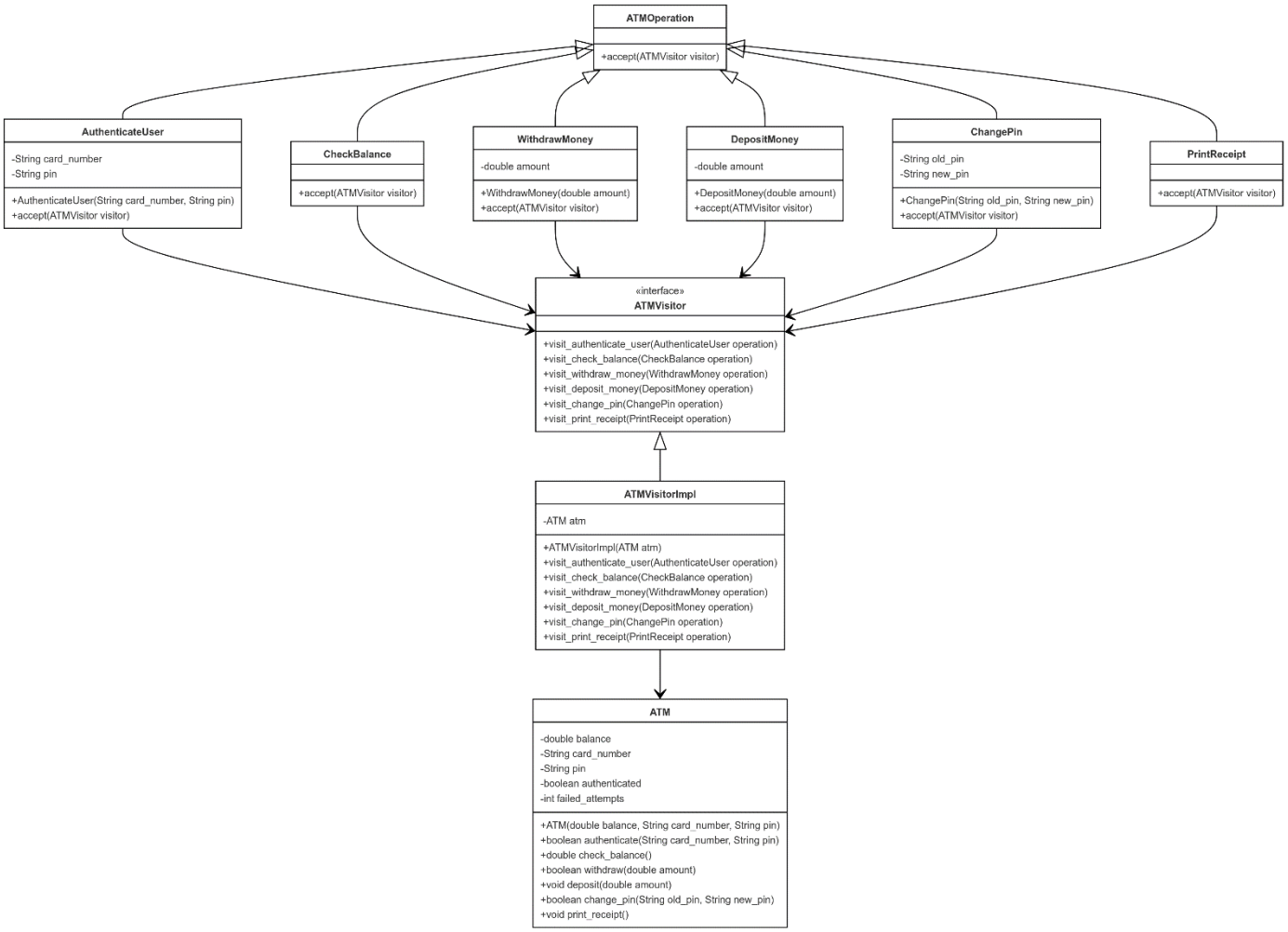

ATMOperation
+execute()
Authentication
-account: Account
-pin: str
-is_authenticated: bool
+execute()
BalanceInquiry
-account: Account
+execute()
Withdrawal
-account: Account
-amount: int
+execute()
Deposit
-account: Account
-amount: int
+execute()
PinChange
-account: Account
-old_pin: str
-new_pin: str
+execute()
PrintReceipt
-transaction_details: str
+execute()
ATMComposite
-operations: List<ATMOperation>
+add_operation(operation: ATMOperation)
+execute()
Account
-account_number: str
-balance: int
-pin: str
-attempts: int
-max_attempts: int
+verify_pin(pin: str) : : bool
+get_balance() : : int
+withdraw(amount: int) : : bool
+deposit(amount: int)
+change_pin(old_pin: str, new_pin: str) : : bool

ผลลัพธ์ของ Python // Visitor Design Pattern

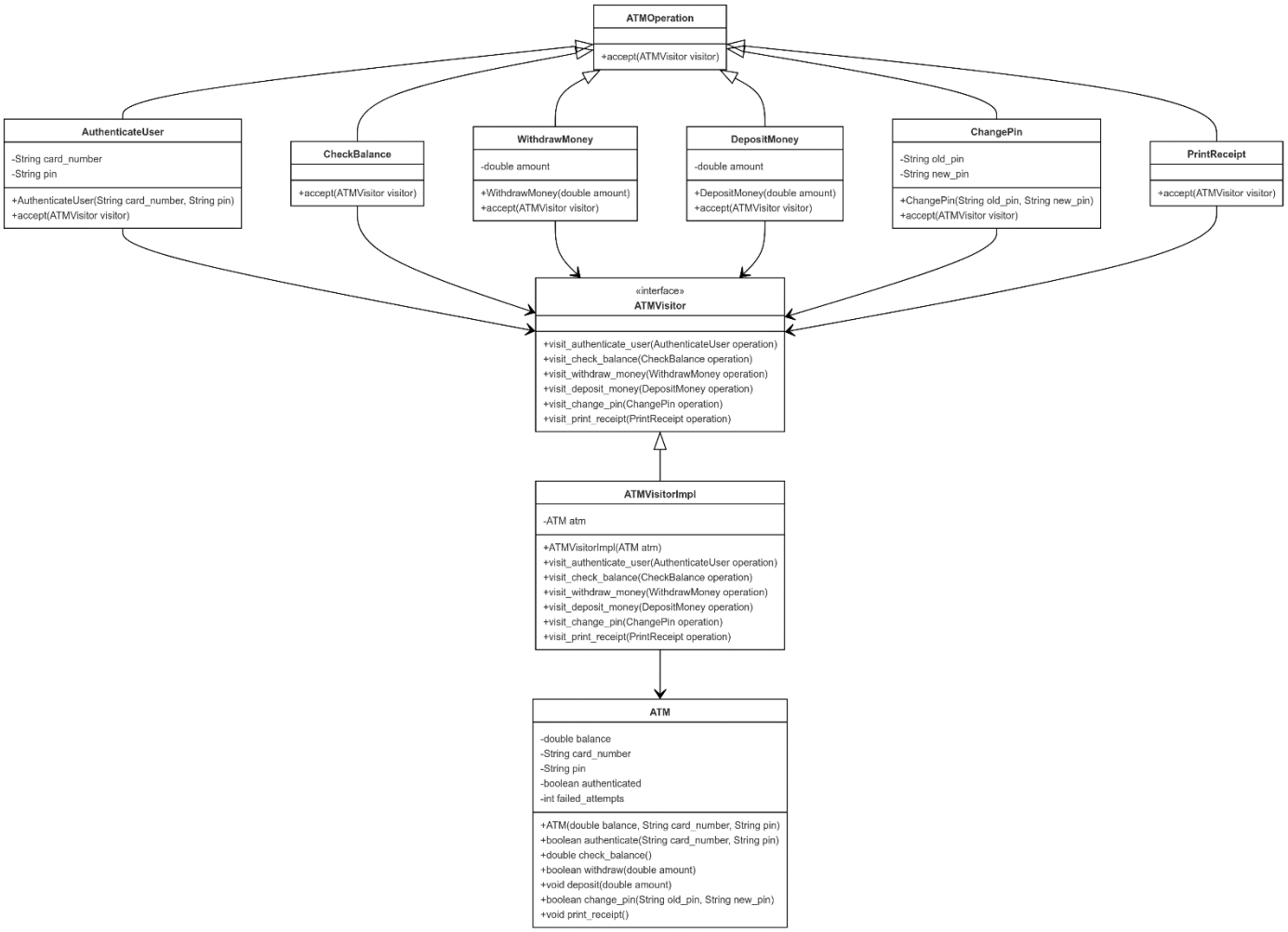
รอบที่1



รอบที่2

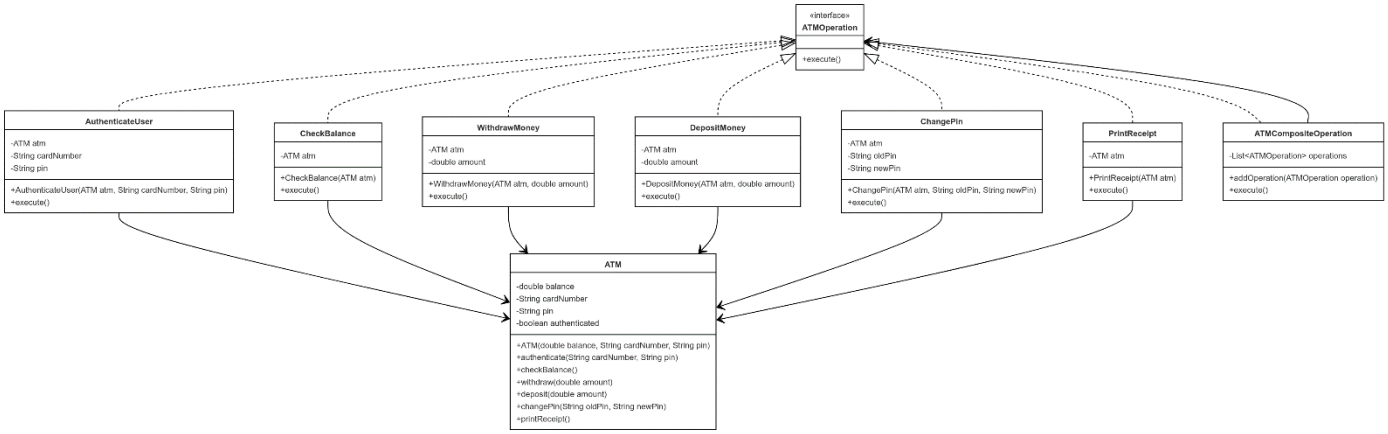

ATMOperation
+accept(ATMVisitor visitor)
AuthenticateUser
-String card_number
-String pin
+AuthenticateUser(String card_number, String pin)
+accept(ATMVisitor visitor)
CheckBalance
+accept(ATMVisitor visitor)
WithdrawMoney
-double amount
+WithdrawMoney(double amount)
+accept(ATMVisitor visitor)
DepositMoney
-double amount
+DepositMoney(double amount)
+accept(ATMVisitor visitor)
ChangePin
-String old_pin
-String new_pin
+ChangePin(String old_pin, String new_pin)
+accept(ATMVisitor visitor)
PrintReceipt
+accept(ATMVisitor visitor)
«interface»
ATMVisitor
+visit_authenticate_user(AuthenticateUser operation)
+visit_check_balance(CheckBalance operation)
+visit_withdraw_money(WithdrawMoney operation)
+visit_deposit_money(DepositMoney operation)
+visit_change_pin(ChangePin operation)
+visit_print_receipt(PrintReceipt operation)
ATMVisitorImpl
-ATM atm
+ATMVisitorImpl(ATM atm)
+visit_authenticate_user(AuthenticateUser operation)
+visit_check_balance(CheckBalance operation)
+visit_withdraw_money(WithdrawMoney operation)
+visit_deposit_money(DepositMoney operation)
+visit_change_pin(ChangePin operation)
+visit_print_receipt(PrintReceipt operation)
ATM
-double balance
-String card_number
-String pin
-boolean authenticated
-int failed_attempts
+ATM(double balance, String card_number, String pin)
+boolean authenticate(String card_number, String pin)
+double check_balance()
+boolean withdraw(double amount)
+void deposit(double amount)
+boolean change_pin(String old_pin, String new_pin)
+void print_receipt()

รอบที่3

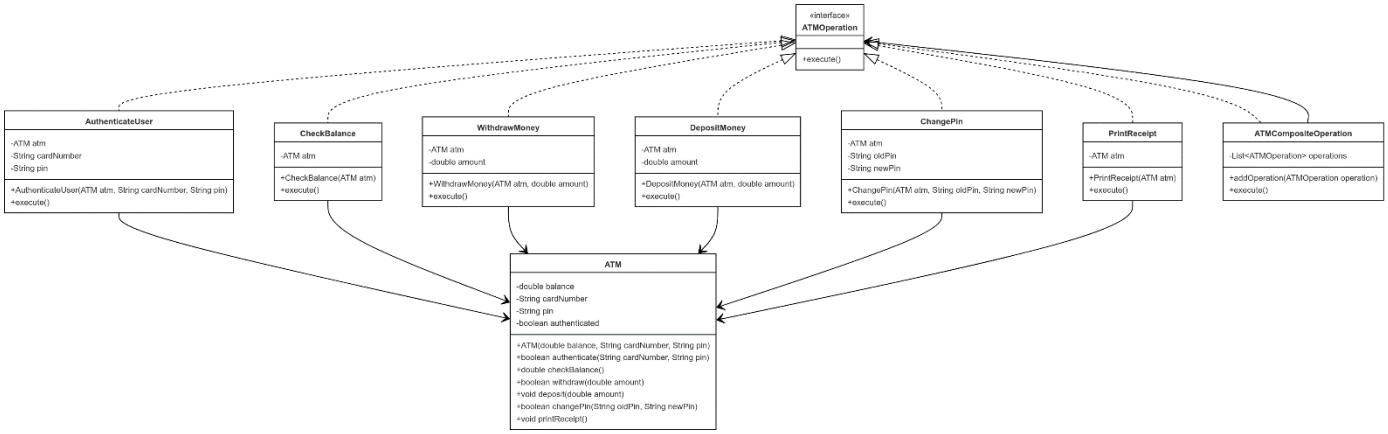

ATMOperation
+accept(ATMVisitor visitor)
AuthenticateUser
-String card_number
-String pin
+AuthenticateUser(String card_number, String pin)
+accept(ATMVisitor visitor)
CheckBalance
+accept(ATMVisitor visitor)
WithdrawMoney
-double amount
+WithdrawMoney(double amount)
+accept(ATMVisitor visitor)
DepositMoney
-double amount
+DepositMoney(double amount)
+accept(ATMVisitor visitor)
ChangePin
-String old_pin
-String new_pin
+ChangePin(String old_pin, String new_pin)
+accept(ATMVisitor visitor)
PrintReceipt
+accept(ATMVisitor visitor)
«interface»
ATMVisitor
+visit_authenticate_user(AuthenticateUser operation)
+visit_check_balance(CheckBalance operation)
+visit_withdraw_money(WithdrawMoney operation)
+visit_deposit_money(DepositMoney operation)
+visit_change_pin(ChangePin operation)
+visit_print_receipt(PrintReceipt operation)
ATMVisitorImpl
-ATM atm
+ATMVisitorImpl(ATM atm)
+visit_authenticate_user(AuthenticateUser operation)
+visit_check_balance(CheckBalance operation)
+visit_withdraw_money(WithdrawMoney operation)
+visit_deposit_money(DepositMoney operation)
+visit_change_pin(ChangePin operation)
+visit_print_receipt(PrintReceipt operation)
ATM
-double balance
-String card_number
-String pin
-boolean authenticated
-int failed_attempts
+ATM(double balance, String card_number, String pin)
+boolean authenticate(String card_number, String pin)
+double check_balance()
+boolean withdraw(double amount)
+void deposit(double amount)
+boolean change_pin(String old_pin, String new_pin)
+void print_receipt()

ผลลัพธ์ของ Java // Composite Design Pattern

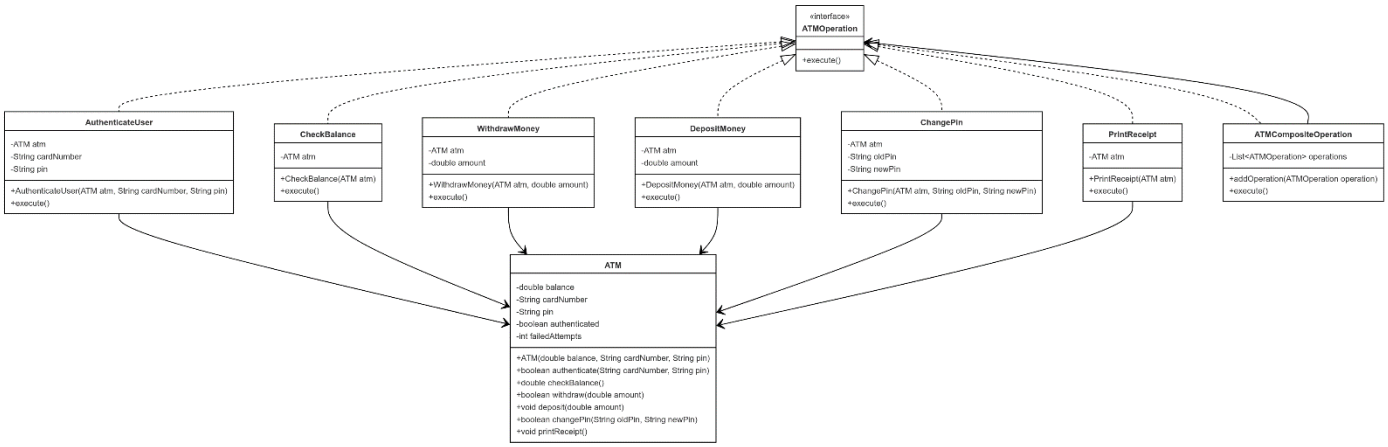
รอบที่1



รอบที่2

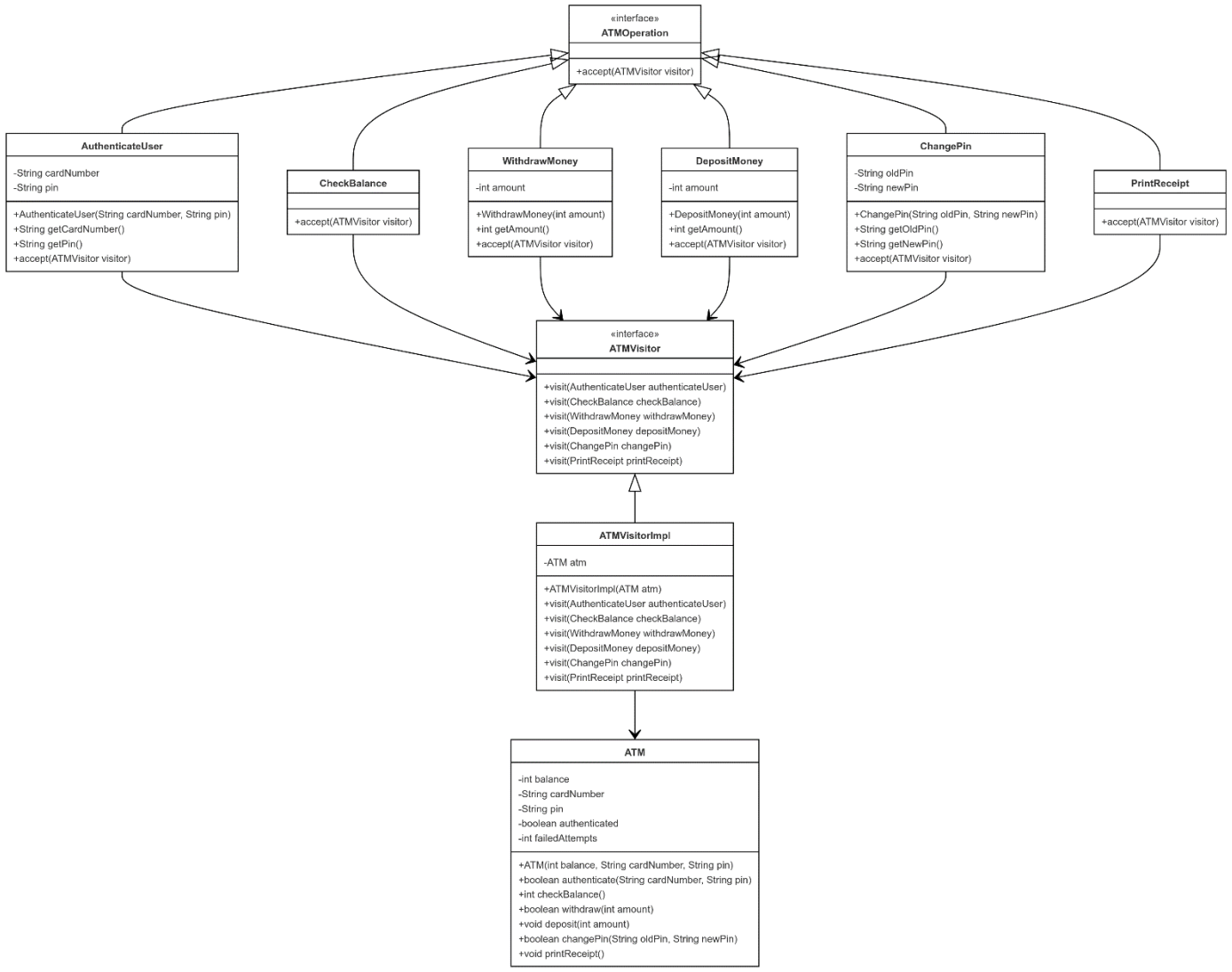


รอบที่3

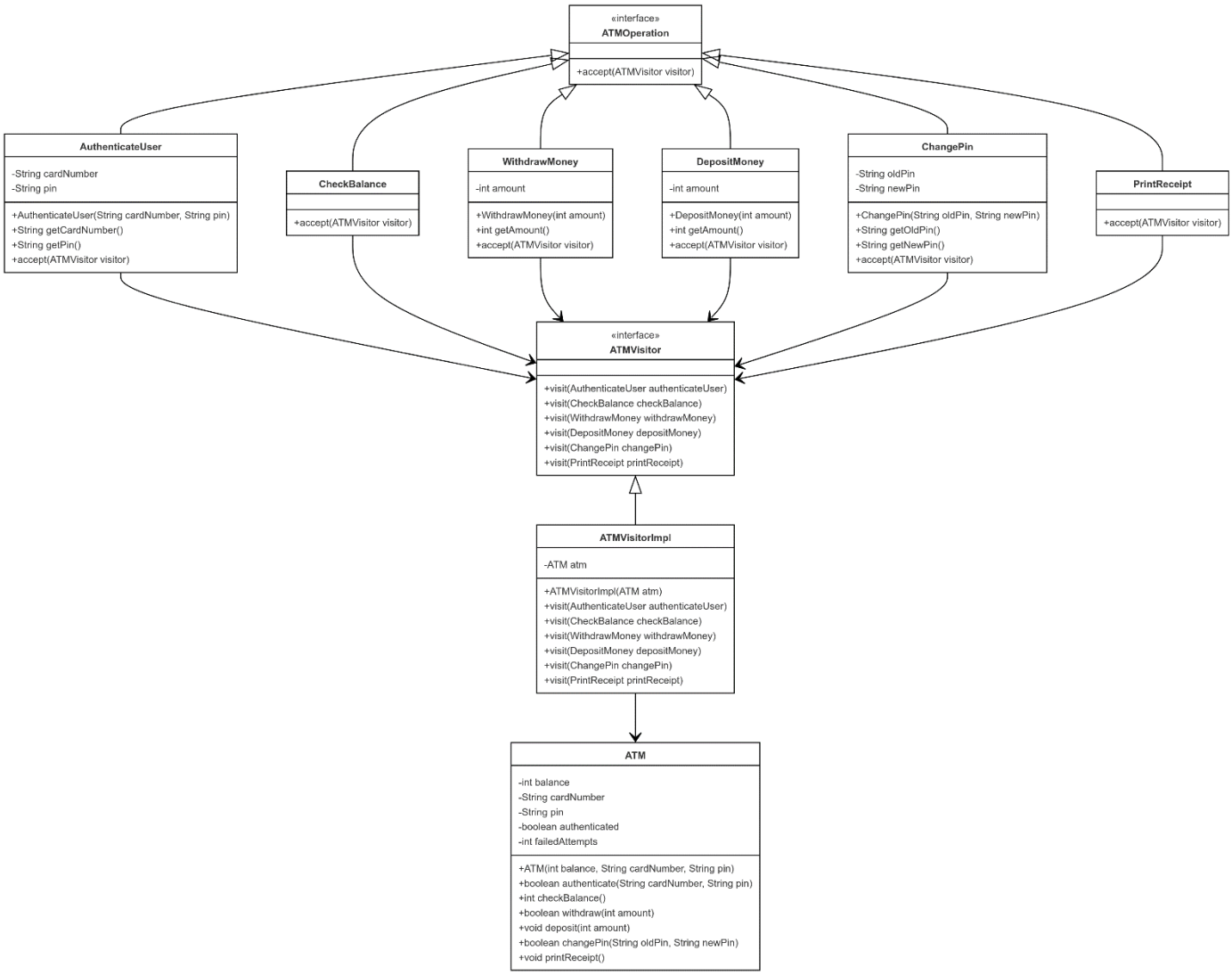


ผลลัพธ์ของ Java // Visitor Design Pattern

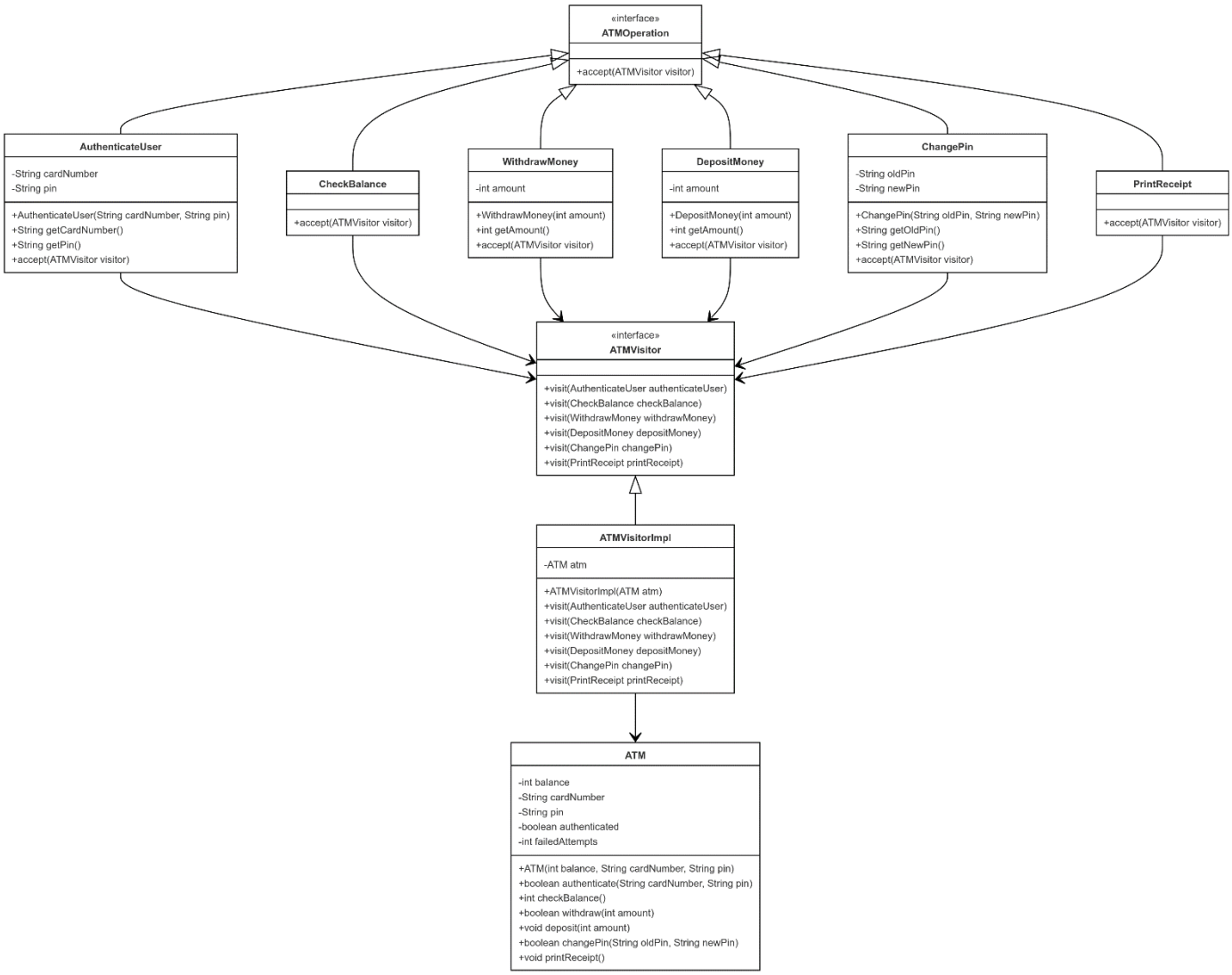
รอบที่1



รอบที่2


«interface»
ATMOperation
+accept(ATMVisitor visitor)
AuthenticateUser
-String cardNumber
-String pin
+AuthenticateUser(String cardNumber, String pin)
+String getCardNumber()
+String getPin()
+accept(ATMVisitor visitor)
CheckBalance
+accept(ATMVisitor visitor)
WithdrawMoney
-int amount
+WithdrawMoney(int amount)
+int getAmount()
+accept(ATMVisitor visitor)
DepositMoney
-int amount
+DepositMoney(int amount)
+int getAmount()
+accept(ATMVisitor visitor)
ChangePin
-String oldPin
-String newPin
+ChangePin(String oldPin, String newPin)
+String getOldPin()
+String getNewPin()
+accept(ATMVisitor visitor)
PrintReceipt
+accept(ATMVisitor visitor)
«interface»
ATMVisitor
+visit(AuthenticateUser authenticateUser)
+visit(CheckBalance checkBalance)
+visit(WithdrawMoney withdrawMoney)
+visit(DepositMoney depositMoney)
+visit(ChangePin changePin)
+visit(PrintReceipt printReceipt)
ATMVisitorImpl
-ATM atm
+ATMVisitorImpl(ATM atm)
+visit(AuthenticateUser authenticateUser)
+visit(CheckBalance checkBalance)
+visit(WithdrawMoney withdrawMoney)
+visit(DepositMoney depositMoney)
+visit(ChangePin changePin)
+visit(PrintReceipt printReceipt)
ATM
-int balance
-String cardNumber
-String pin
-boolean authenticated
-int failedAttempts
+ATM(int balance, String cardNumber, String pin)
+boolean authenticate(String cardNumber, String pin)
+int checkBalance()
+boolean withdraw(int amount)
+void deposit(int amount)
+boolean changePin(String oldPin, String newPin)
+void printReceipt()

รอบที่3



วิเคราะห์ผลการทดลอง copilot

จากผลลัพธ์พบว่า ในด้านภาษา programming นั้น copilot เขียนออกมาได้อย่างถูกต้องและแม่นยำทั้ง python และ java แต่ทุกครั้งที่ใส่ prompt code ที่ออกมาแต่ละครั้งจะมีการเปลี่ยนแปลงนิดหน่อย เป็นการปรับและเปลี่ยนแปลงให้ดีขึ้นจาก prompt ก่อนๆ ส่วนในด้าน Testing นั้น code ที่ออกมาจะไม่ค่อยมีความแตกต่างจาก code ก่อนหน้าแต่ทำการ test ได้ปกติถึงจะมี test fail ในบางครั้งก็ตาม

โดยสรุป โมเดล AI copilot ถือว่าเป็นเครื่องมือที่มีประสิทธิภาพเป็นอย่างมากในการช่วยเขียนโคด และยังสามารถให้คำแนะนำมวิธีใช้และแก้ข้อผิดพลาดเพื่อเพิ่มประสิทธิภาพในการเขียนโคดได้อีกด้วย แต่ผู้ใช้ก็ควรใช้ copilot เป็นผู้ช่วยในการเขียนแทนที่จะเป็นตัวหลักเพราะ copilot ก็ไม่ได้แม่นยำ 100% และการตัดสินใจหลักควรอยู่ในมือของผู้พัฒนา