

Composite & Visitor

Presented by Group 8

รายชื่อสมาชิก

1.นายธีรัช เจริญวารี	653380268-0	sec.1
2.นายกันแพงเพชร สิงห์บรรณ	653380120-2	sec.1
3.นายณัทส ประสงค์ดี	653380128-6	sec.1

Composite Design Pattern

Composite Design Pattern คืออะไร

Composite Design Pattern เป็นหนึ่งในรูปแบบการออกแบบเชิงวัตถุที่ใช้ในการจัดการโครงสร้างข้อมูลที่มีความสัมพันธ์แบบลำดับชั้น โดยที่วัตถุที่อยู่ในโครงสร้างนั้นสามารถมีโครงสร้างย่อยๆ ได้เป็น Tree Structure หลักการสำคัญของ Composite Pattern คือการกำหนดให้กั้งวัตถุเดียว (Leaf) และกลุ่มของวัตถุ (Composite) สามารถใช้งานในลักษณะเดียวกัน โดยทั่วไปจะมีอินเทอร์เฟซร่วม (Interface หรือ Abstract Class) ที่ให้กั้งวัตถุเดียวและกลุ่มวัตถุทำงาน ทำให้สามารถจัดการโครงสร้างข้อมูลได้อย่างยืดหยุ่น และง่ายต่อการใช้งาน

หลักการทำงาน

- Component: อินเตอร์เฟซหรือคลาสฐานที่กำหนดการกระทำที่สามารถทำได้กับอ็อบเจ็กต์กั้งหมดในโครงสร้าง
- Leaf: คลาสที่แทนอ็อบเจ็กต์ที่ไม่สามารถมีลูก (child) ได้ เช่น ไฟล์
- Composite: คลาสที่แทนอ็อบเจ็กต์ที่สามารถมีลูกได้ เช่น โฟลเดอร์ ซึ่งสามารถมี Leaf หรือ Composite อื่นๆ เป็นลูกได้

สรุป

ช่วยให้การทำงานกับโครงสร้างข้อมูลที่มีลักษณะเป็นต้นไม้มีความยืดหยุ่นและมีประสิทธิภาพมากขึ้น

Visitor Design Pattern

Visitor Design Pattern คืออะไร

Visitor Design Pattern เป็นหนึ่งในรูปแบบการออกแบบเชิงวัตถุที่ใช้ในการเพิ่มฟังก์ชันการทำงานให้กับวัตถุโดยไม่ทำการแก้ไขคลาสของวัตถุนั้นโดยตรง ซึ่งมีประโยชน์ในการแยกตัว域การดำเนินงานออกจากโครงสร้างของข้อมูล เพื่อให้สามารถเพิ่มฟังก์ชันใหม่ๆ ได้ง่ายขึ้น

หลักการสำคัญของ Visitor Pattern คือการใช้ Visitor เป็นตัวแทนในการดำเนินการบางอย่างบนวัตถุ และวัตถุที่ต้องการให้มีการดำเนินการเหล่านั้นจะต้องยอมรับการเข้ามาของ Visitor โดยการเรียกใช้เมธอด accept() ซึ่งจะทำให้ Visitor สามารถทำงานบนวัตถุเหล่านั้นได้

หลักการทำงาน

- Visitor: คลาสที่มีเมธอดต่างๆ สำหรับการดำเนินการกับอ็อบเจกต์แต่ละประเภท
- Element: อินเตอร์เฟซหรือคลาสฐานที่กำหนดเมธอด accept ซึ่งใช้ในการรับ visitor
- ConcreteElement: คลาสที่ใช้งานจริง ซึ่งจะมีการกำหนดการทำงานเฉพาะเมื่อ visitor เข้ามา
- Object Structure: โครงสร้างที่เก็บอ็อบเจกต์ประเภทต่างๆ และอนุญาตให้ visitor เข้ามายืนยันได้

สรุป

Visitor Design Pattern เป็นเครื่องมือที่มีประโยชน์ในการจัดการกับการดำเนินการต่างๆ บนโครงสร้างอ็อบเจกต์ โดยทำให้สามารถเพิ่มฟังก์ชันใหม่ได้ง่ายและไม่ต้องปรับเปลี่ยนโครงสร้างเดิม

Cyclomatic Complexity

Cyclomatic Complexity เป็นตัวชี้วัดทางซอฟต์แวร์ที่ใช้วัดความซับซ้อนของโค้ด โดยพิจารณาจากจำนวนเส้นทางที่เป็นไปได้ ในการรันโปรแกรม เป็นแนวคิดที่ช่วยประเมินความยากง่ายในการทำความเข้าใจและทดสอบโค้ด โดยเฉพาะในกรณีของฟังก์ชัน เมธอด หรือโมดูล

วิธีการคำนวณ Cyclomatic Complexity:

ความซับซ้อนเชิงไฮโคลมาติกสามารถคำนวณได้จาก Control Flow Graph (CFG) ของโปรแกรม โดยที่:

- Nodes (จุด): แทนสถานะหรือคำสั่งของโปรแกรม
- Edges (เส้นทาง): แทนการไหลของโปรแกรมจากคำสั่งหนึ่งไปยังอีกคำสั่งหนึ่ง

สูตรในการคำนวณ:

$$\text{Cyclomatic Complexity} = E - N + 2P$$

- E คือ จำนวนเส้นทาง (Edges)
- N คือ จำนวนจุด (Nodes)
- P คือ จำนวนส่วนที่เชื่อมต่อ กัน

ในโค้ดที่มีเงื่อนไขการตัดสินใจ เช่น คำสั่ง if, while, for, switch-case แต่ละคำสั่งเหล่านี้จะเพิ่มเส้นทางใหม่ให้โปรแกรม ซึ่งจะทำให้ค่า Cyclomatic Complexity สูงขึ้น

ความหมายของค่า Cyclomatic Complexity:

- 1-10: ความซับซ้อนต่ำ, โปรแกรมมีเส้นทางที่เข้าใจง่ายและทดสอบง่าย
- 11-20: ความซับซ้อนปานกลาง, การทดสอบเริ่มยากขึ้น
- 21-50: ความซับซ้อนสูง, ต้องระวังในการบำรุงรักษาและทดสอบ
- มากกว่า 50: ความซับซ้อนมาก, อาจต้องพิจารณาออกแบบโค้ดใหม่

REQUIREMENT

จำลองการทำงานระบบบัญชี ATM ที่สามารถเลือกฝากถอน
เช็คยอดเงินในบัญชี เปลี่ยนรหัสผ่านบัญชี และพิมพ์ใบเสร็จ
โดยกำหนดให้ Generative AI ทั้ง 4 Model ออกแบบซอฟต์แวร์
และโค้ดทดสอบ โดยซอฟต์แวร์ต้องออกแบบตามดีไซน์
แพทเทิร์น Composite & Visitor

PYTHON CHAT-GPT 40



chat-gpt 40 สามารถสร้าง โค้ด
ออกแบบได้ถูกต้องตาม design
pattern ที่กำหนด ไม่มีข้อผิดพลาด
ที่ต้องแก้ไข ใช้เครื่องมือ pylint ใน
การตรวจสอบ

CODE ABILITY



chat-gpt 40 สามารถทำคะแนน
complexity โดยเฉลี่ยในระดับ
ฟังก์ชันมีค่าความซับซ้อนไม่เกิน 4
ซึ่งอยู่ในเกณฑ์ที่ยอมรับได้ และทำ
คะแนน maintainability ออกแบบ
เฉลี่ย 60% ในส่วนของชอล์โค้ด
และ 40 % ในส่วนเทล์โค้ด

SOFWARE MATRIC



ในรอบที่ 3 ของทั้งสอง design
pattern chat-gpt 40 สามารถ
สร้างเทล์โค้ด ที่มี 100% Branch
and statements coverage วัดโดย
ใช้ pytest -cov

TEST QUAILITY

JAVA CHAT-GPT 40



chat-gpt 40 สามารถสร้าง โค้ด
ออกแบบได้ถูกต้องตาม design
pattern ที่กำหนด ในโค้ดทดสอบ
มีส่วนที่ต้องแก้ไขคือการใช้ library
JUnit เวอร์ชัน 4 ซึ่งมีโค้ดทดสอบ
ในไฟล์อื่นใช้เวอร์ชัน 5 ทำให้
เวอร์ชัน library ไม่เป็นไปในทาง
เดียวกัน

CODE ABILITY



chat-gpt 40 สามารถทำคะแนน
complexity โดยเฉลี่ยในระดับ
ฟังก์ชันมีค่าความซับซ้อนไม่เกิน 3
ซึ่งอยู่ในเกณฑ์ที่ยอมรับได้ วัดจาก
เครื่องมือ Codalyze - Code
Complexity Report Generator

SOFWARE MATRIC



ในรอบที่ 3 ของ Visitor design
pattern Chat-GPT 40 สามารถ
สร้างเทลส์โค้ดที่มี Branch
coverage 90%
statements coverage 99%
วัดโดยใช้ JaCoCo

TEST QUAILITY

Gemini 1.5 Flash **VS** Gemini 1.5 Pro

ในรุ่น Flash เป็น Model กี่เน้นความรวดเร็วในการสร้างคำ แต่ก็แลกมาด้วยความแม่นยำกี่ต่ำลงมาทำให้โค้ดที่ได้อาจไม่ครอบคลุมกับ Requirement ทั้งหมดที่ได้ให้ไป

ส่วนในรุ่น Pro เป็น Model กี่เน้นความแม่นยามากกว่า แต่ก็จะใช้เวลาและกินทรัพยากรมากกว่าตัว Flash ตามไปด้วย และผลลัพธ์ที่ได้ก็มีความแม่นยำเพิ่มมากขึ้นและสามารถนำไปประยุกต์ใช้ได้ดีมากยิ่งขึ้น

Prompt Example

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.

Gemini 1.5 Flash

Gemini 1.5 Flash สามารถสร้างโค้ดถูกต้องตาม หลัก Composite และ Visitor Design Pattern แต่รายละเอียดการทำงานของฟังก์ชันนี้จะไม่ซับซ้อน และ ในบางกรณีที่ฟังก์ชันจะมีการทำงานที่ไม่ตรงกับ Requirement เห็นได้ชัดใน Python ดังเช่นตัวอย่าง

Python Composite / Round 1

```
class PrintReceipt(ATMComponent):
    def process(self, user):
        print("Printing receipt...")
```

ในส่วน Java มีข้อพิດพลาดน้อยกว่า

Gemini 1.5 Pro

Gemini 1.5 Pro สามารถสร้างโค้ด ได้ตามหลัก Composite และ Visitor Design Pattern และครอบคลุมฟังก์ชันหลักที่กำหนด ในส่วนโค้ดที่ให้มีจุดผิดพลาดน้อยและส่วนมากจะสามารถใช้งานได้กันที

โค้ดจาก Gemini 1.5 Pro มีค่า Cyclomatic Complexity เวลี่ยที่ไม่เกิน 3

ตัวอย่างค่า MI (Maintainability Index) จาก Gemini 1.5 Pro / Python / Visitor

Round	1	2	3
code.py	50.74	50.47	76.71
unit_test.py	75.04	64.47	60.5

GitHub Copilot

Python programming language

GitHub Copilot เจน code Python ได้มาครบก่อนและไม่พบ error แต่ต้อง testing กลับพบได้ว่ามีความแม่นยำ 80% ค่า Cyclomatic Complexity บันทึกได้น้อยกว่า 10 ทั้งหมดไม่ว่าจะเป็น Composite หรือ Visitor ก็ตามแปลงว่า code บันทึกความซับซ้อนในการเขียน code ของ GitHub Copilot ต่ำและมีความแม่นยำสูงจึงทำให้ง่ายต่อการถูแร็คษา ถูกใจจากตัวอย่าง Composite/UnitTest3 และ Visitor/UnitTest3

Composite

Type	Line	Class/Method	Grade	Score
F	18	test_authentication_max_attempts	A	4
F	8	test_authentication_success	A	2
F	13	test_authentication_failure	A	2
F	25	test_balance_inquiry	A	2
F	30	test_withdrawal_success	A	2
		test_withdrawal_insufficient_funds		
F	35	nd	A	2
F	40	test_deposit	A	2
F	45	test_pin_change_success	A	2
F	50	test_pin_change_failure	A	2
F	5	account	A	1
F	55	test_print_receipt	A	1
F	61	test_atm_composite	A	1

Visitor

Type	Line	Class/Method	Grade	Score
F	12	test_authenticate_user	A	3
F	27	test_withdraw_money	A	3
F	43	test_change_pin	A	3
F	21	test_check_balance	A	2
F	37	test_deposit_money	A	2
F	5	atm	A	1
F	9	visitor	A	1
F	53	test_print_receipt	A	1

GitHub Copilot

Java programming language

GitHub Copilot เจน code Java ได้มาครบถ้วนและไม่พบ error แต่ต้อง testing กลับพบได้ว่ามีความแม่นยำ 86% ส่วนค่า Cyclomatic Complexity ได้เหมือนกับตอนเจน python คือได้น้อยกว่า 10 กึ่งหมุด

4	Composite					
5	Round1					
6	ATM.java					
7	Function Name	Start Line	End Line	Cyclomatic Complexity (Threshold: 10)	Lines of Code (Threshold: 50)	Parameter Count (Threshold: 4)
8	ATM::ATM	10	16	1	7	3
9	ATM::authenticate	18	34	4	16	2
10	ATM::checkBalance	36	44	2	9	0
11	ATM::withdraw	46	60	3	15	1
12	ATM::deposit	62	69	2	8	1
13	ATM::changePin	71	85	3	15	2
14	ATM::printReceipt	87	94	2	7	0

**Thank you
very much!**