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

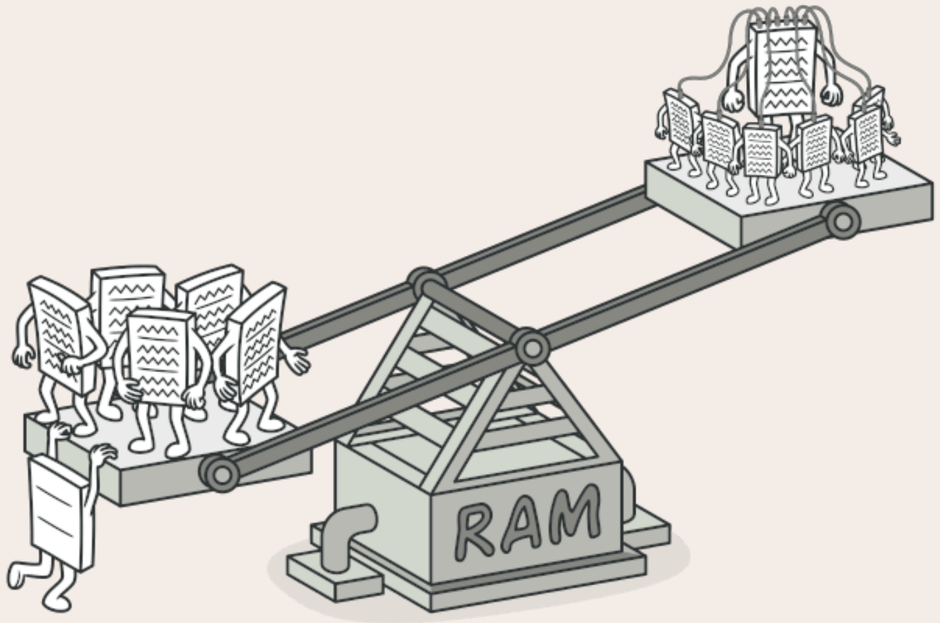
Flyweight Pattern & Interpreter pattern

เครื่องมือ AI-Assisted ที่ใช้

- Chatgpt 4.0
- Gemini 1.5 Flash
- Gemini 1.5 Pro
- Github Copilot

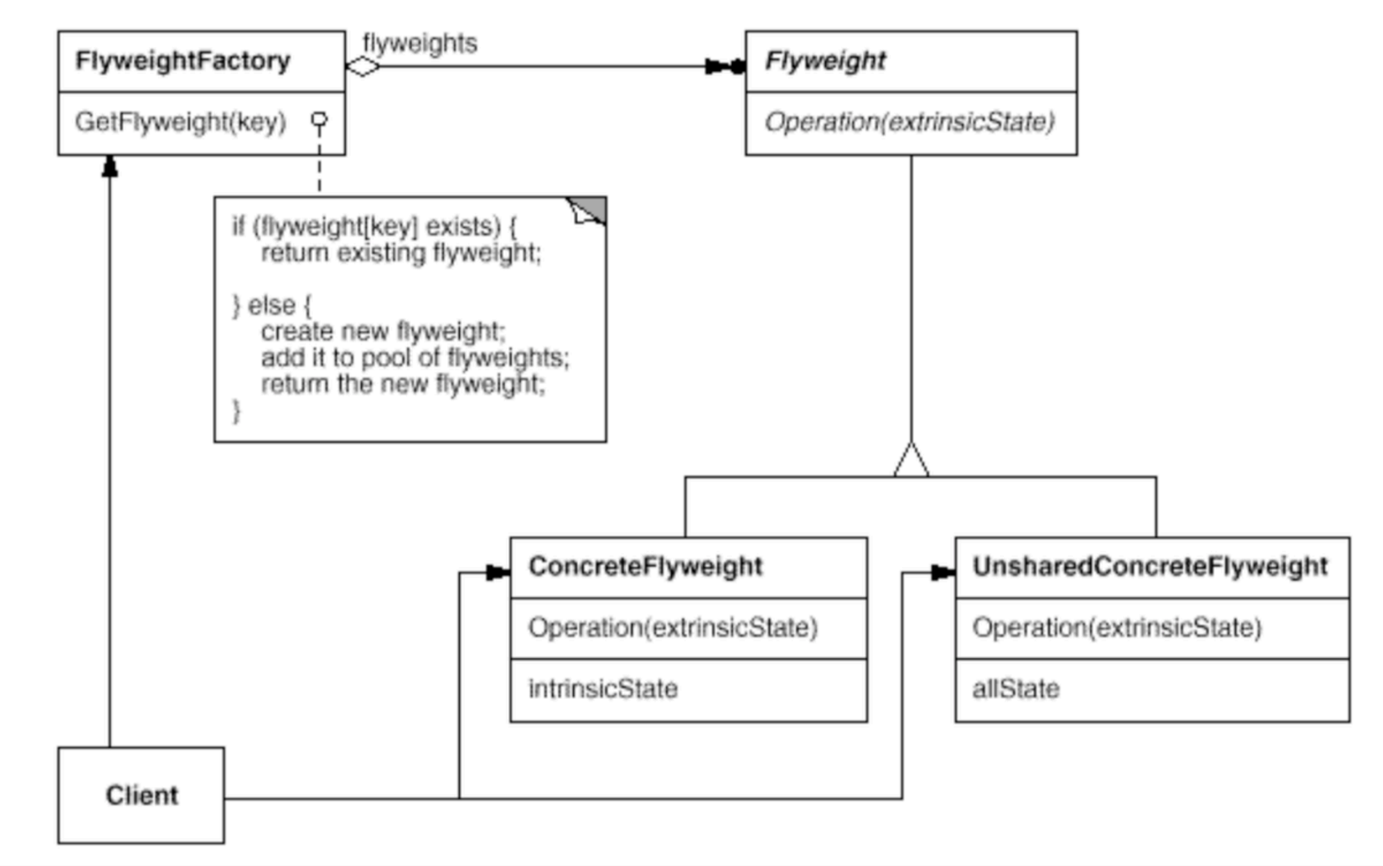
Flyweight

เป็นเทคนิคที่ช่วยลดการใช้หน่วยความจำและเพิ่มประสิทธิภาพของแอปพลิเคชัน โดยการแชร์วัตถุที่เหมือนกันแทนที่จะสร้างวัตถุใหม่ทุกครั้ง โดยจะแยกแยะระหว่างสถานะภายในของวัตถุ (intrinsic state) ซึ่งเหมือนกันในทุก instance และสถานะภายนอก (extrinsic state) ซึ่งสามารถแตกต่างกันได้ระหว่างวัตถุ



ที่มา: https://refactoring.guru/design-patterns/flyweight

Structure



ที่มา: https://itnext.io/easy-patterns-flyweight-dab4c018f7f5?gi=1624f3aa566d

Flyweight : ประกาศ interface ที่ให้ Flyweights สามารถรับและดำเนินการกับสถานะภายนอก (extrinsic state)

ConcreteFlyweight : การเขียน code เพื่อใช้งานตาม interface ของ Flyweight สถานะภายในไว้ โดยสถานะภายในจะเป็นข้อมูลที่ไม่ขึ้นอยู่กับบริบทของการใช้งาน และสามารถแชร์กันได้

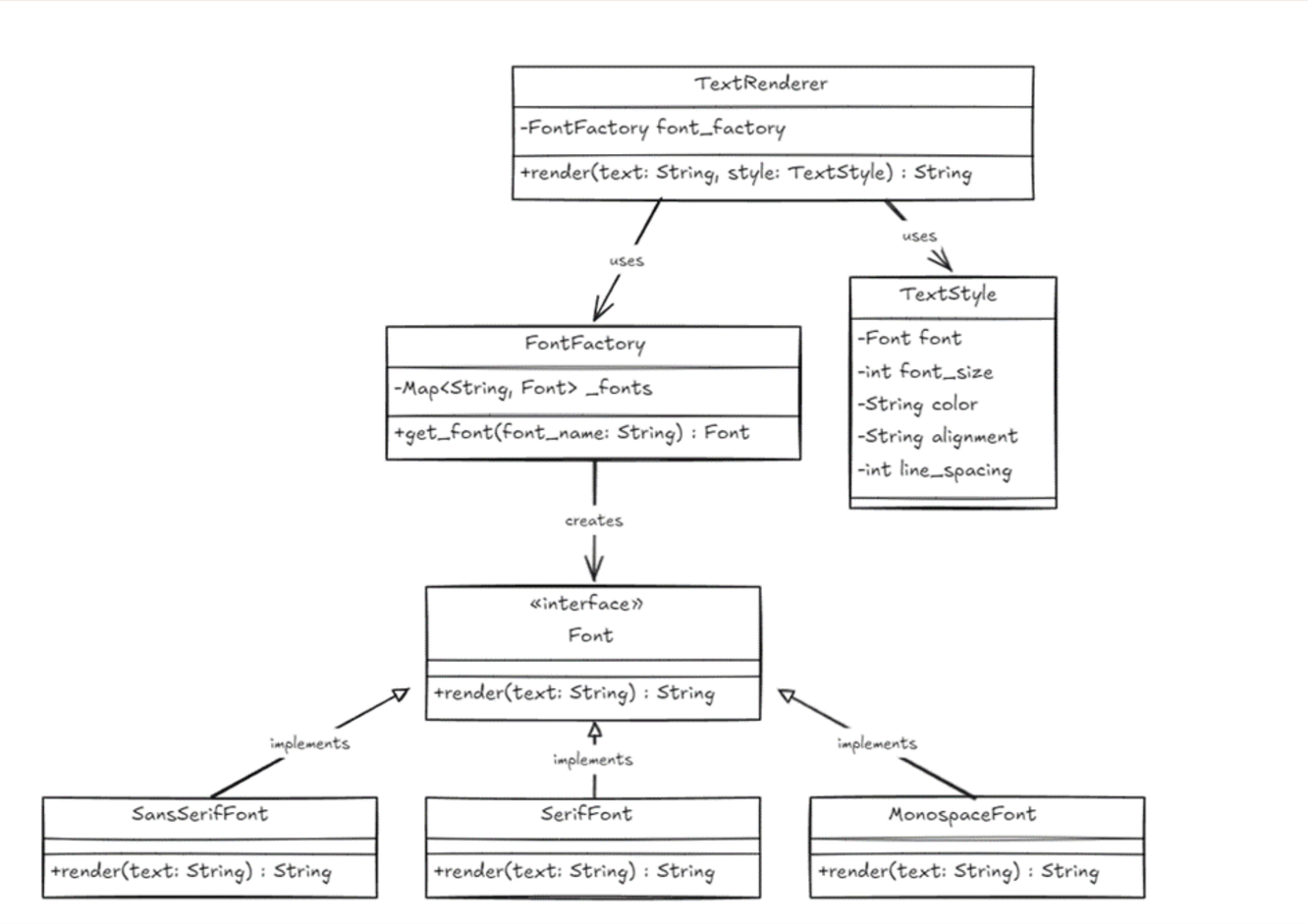
UnsharedConcreteFlyweight : เป็นคลาสที่ใช้เพื่อสร้างวัตถุที่ไม่สามารถแชร์ได้ มักจะใช้สำหรับข้อมูลเฉพาะที่ไม่สามารถแชร์ระหว่างวัตถุอื่น ๆ

FlyweightFactory : ทำหน้าที่จัดการ object ประเภท Flyweight และมั่นใจว่าจะเกิดการแชร์ในการใช้ object ร่วมกันอย่างเหมาะสม เมื่อ Client request การใช้งานเข้ามา Flyweight Factory นี้จะดูว่ามีอยู่แล้วหรือไม่ ถ้ายังไม่มีก็จะสร้างใหม่ออกไปและจัดเก็บไว้ใน Factory อย่างถูกต้องได้

Client: ส่วนที่เรียกใช้ Flyweight Factory เพื่อเข้าถึง Object ประเภท Flyweight โดยมีหน้าที่รับผิดชอบเรื่องการจัดการสถานะภายนอก ที่เกี่ยวข้องกับ Flyweight แต่ละตัว (ซึ่งมีโอกาสแตกต่างกันได้ และแชร์กันไม่ได้)

Use case ตัวอย่าง

ในตัวอย่างคลาสสิกของการใช้ Flyweight Pattern คือ Text Rendering System ซึ่งต้องจัดการกับการแสดงผลตัวอักษรหลายตัวที่มีลักษณะเฉพาะตัว เช่น ขนาด (fontSize), การจัดเรียง (Alignment), และสี (Color) ของตัวอักษรแต่ละตัว



1. TextRenderer
   - เป็นคลาสหลักที่รับผิดชอบการแสดงข้อความ โดยมีการใช้ FontFactory เพื่อสร้างหรือดึงฟอนต์มาใช้งานร่วมกับ TextStyle (เช่น ฟอนต์, ขนาด, สี, การจัดวาง)

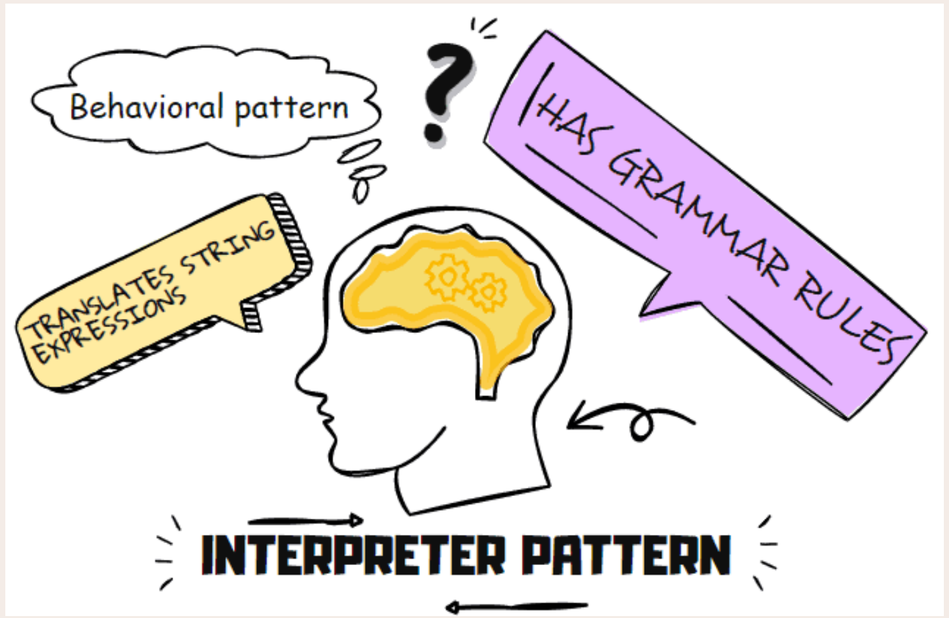
2. FontFactory
   - Method get_font() จะรับชื่อของฟอนต์มา ถ้าฟอนต์ยังไม่ถูกสร้างขึ้นมาก่อน ก็จะสร้างฟอนต์ใหม่ และเก็บไว้ใน Map เพื่อใช้ในครั้งต่อไป
3. Font (Interface)
   - เป็นอินเตอร์เฟซหลักสำหรับฟอนต์ โดยมีเมธอด render() ที่รับข้อความและส่งผลลัพธ์ออกมาเป็น string

4. SansSerifFont, SerifFont, MonospaceFont (Concrete Flyweights)
   - คลาสที่สืบทอดจากอินเตอร์เฟซ Font และเป็นตัวแทนของประเภทฟอนต์ที่แตกต่างกัน

5. TextStyle
   - คลาสนี้เก็บรายละเอียดของสไตล์ของข้อความ เช่น ฟอนต์, ขนาด, สี, การจัดวาง, และการจัดระยะบรรทัด

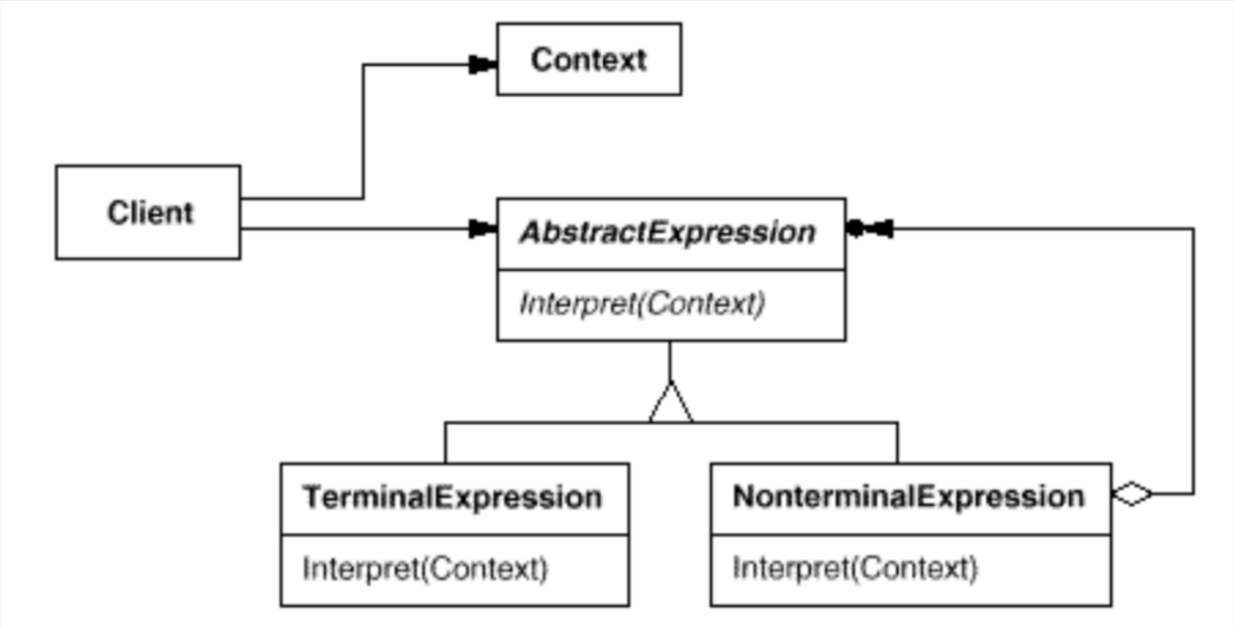
Interpreter

เป็นแพทเทิร์นที่ใช้ในการสร้างระบบที่สามารถตีความและประมวลผลภาษาเฉพาะ (domain-specific languages) หรือกฎที่กำหนดไว้ โดยมักใช้ในการพัฒนาโปรแกรมที่ต้องทำงานกับข้อมูลที่มีโครงสร้างหรือรูปแบบเฉพาะเจาะจง



ที่มา : https://methodpoet.com/interpreter-pattern/

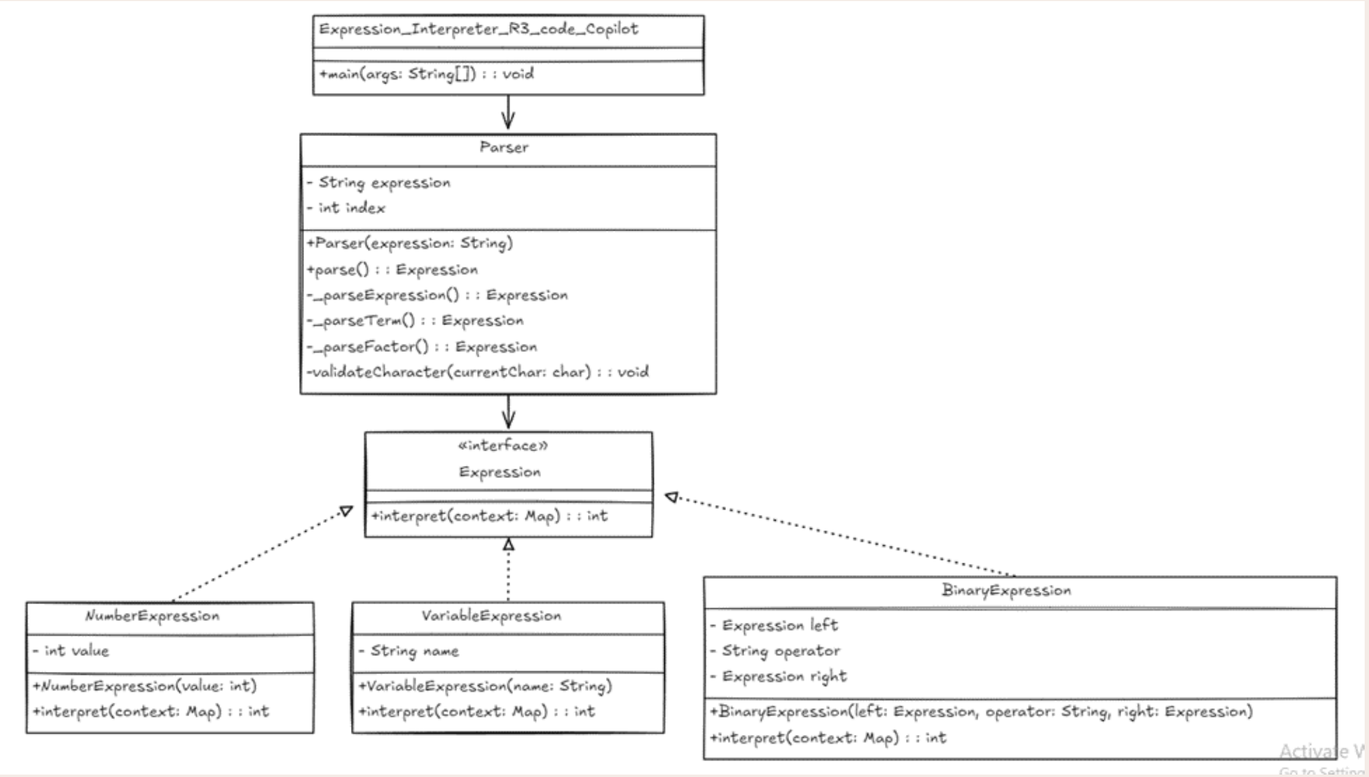
Structure



1. **Client**
   - เป็นผู้ส่งปัญหาหรือภาษาที่ต้องการให้ตีความพร้อมกับบริบท (Context) ที่ต้องใช้ในการตีความ
2. **Context**
   - เก็บข้อมูลสถานะปัจจุบันหรือตัวแปรที่ต้องใช้ในการตีความนิพจน์ เช่น ค่าตัวแปรหรือข้อมูลที่จำเป็น
3. **AbstractExpression**
   - ทำหน้าที่เป็นคลาสฐานสำหรับนิพจน์ทุกประเภท โดยบังคับให้นิพจน์ทุกประเภทมีเมธอด interpret() เพื่อใช้ในการตีความ
4. **TerminalExpression**
   - ใช้ตีความข้อมูลที่เป็นรูปแบบพื้นฐานที่สุด (เช่น ตัวเลข) ที่ไม่ต้องพึ่งพานิพจน์อื่น
5. **NonterminalExpression**
   - ใช้ตีความนิพจน์ที่ซับซ้อน เช่น การคำนวณหรือการดำเนินการทางคณิตศาสตร์ ซึ่งรวมเอาผลลัพธ์จากนิพจน์ย่อยต่าง ๆ

Use case ตัวอย่าง

ในตัวอย่างคลาสสิกของการใช้ Interpreter Pattern คือ การตีความนิพจ์ทางคณิตศาสตร์ โดยมีการแยกส่วนการตีความของแต่ละองค์ประกอบออกเป็นคลาสย่อย



1. Expression (Interface)
   - เป็นคลาสหลักที่เป็นตัวแทนของนิพจน์ต่าง ๆ ทุกคลาสย่อยที่เป็นนิพจน์จะต้องสืบทอดจากคลาสนี้

2. Parser
   - แยกส่วนประกอบของนิพจน์เพื่อส่งต่อให้คลาสอื่น ๆ ตีความ

3. NumberExpression (Terminal Expression)
   - ใช้แทนนิพจน์ที่เป็นตัวเลขหรือตัวแปรที่สามารถตีความได้โดยตรง

4. VariableExpression (Terminal Expression)
   - เป็นนิพจน์ที่ใช้แทนตัวแปรที่ผู้ใช้ระบุ
   - ทำการตีความโดยอ้างอิงถึง context ซึ่งเก็บข้อมูลของตัวแปรและค่าที่แท้จริง

5. BinaryExpression (Non-terminal Expression)
   - ใช้แทนนิพจน์แบบ Binary เช่น การบวก ลบ คูณ หาร
   - คลาสนี้มีการเก็บนิพจน์ทางซ้ายและขวา พร้อมกับตัวดำเนินการ (operator) เช่น "+" หรือ "-"
   - ในการตีความ จะตีความทั้งฝั่งซ้ายและขวา และนำผลมาประมวลผลตามตัวดำเนินการที่กำหนด

วิธีการดำเนินงานทดสอบ

1.กำหนดพฤติกรรมที่ต้องการของคลาส/ฟังก์ชัน เช่น ใน Flyweight เราต้องการระบบแสดงผลข้อความ (Text Rendering System) ที่สามารถจัดการฟอนต์ จัดการสีข้อความ
ถ้าใน Interpreter เราต้องการระบบประเมินค่านิพจน์ทางคณิตศาสตร์ ที่รองรับการประเมินค่าของนิพจน์พื้นฐาน เช่น การบวก, ลบ, คูณ, และหาร เป็นต้น

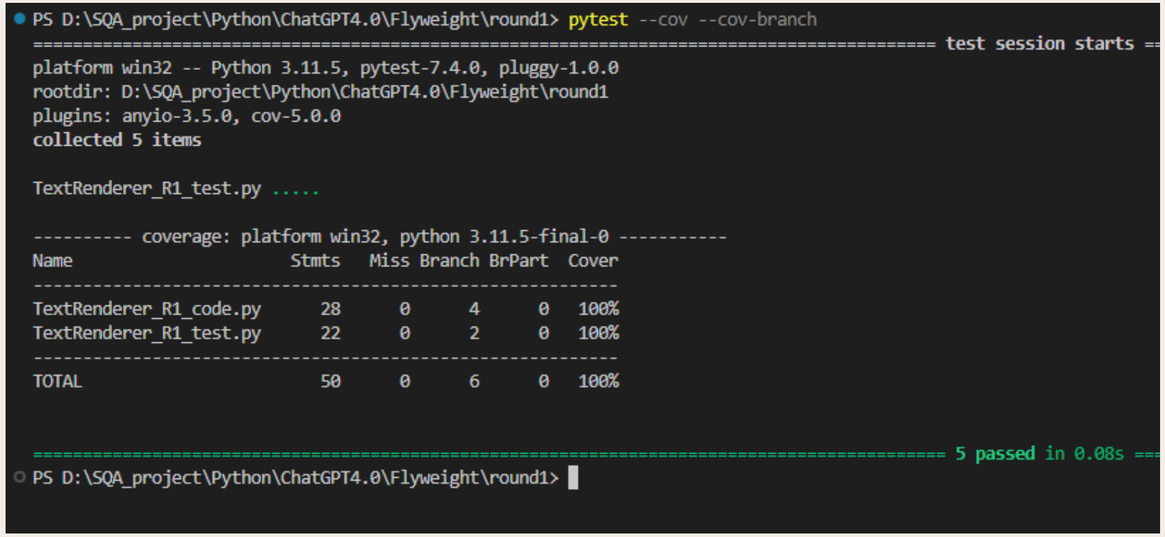
2.ออกแบบ prompt เพื่อให้ AI ใช้ในการ generate โค๊ด และ Test Code

3.แก้ไขโค้ดในกรณีที่โค๊ดรันไม่ได้

4.วัดคุณภาพของ Code และ Test Code ด้วย Codalyze

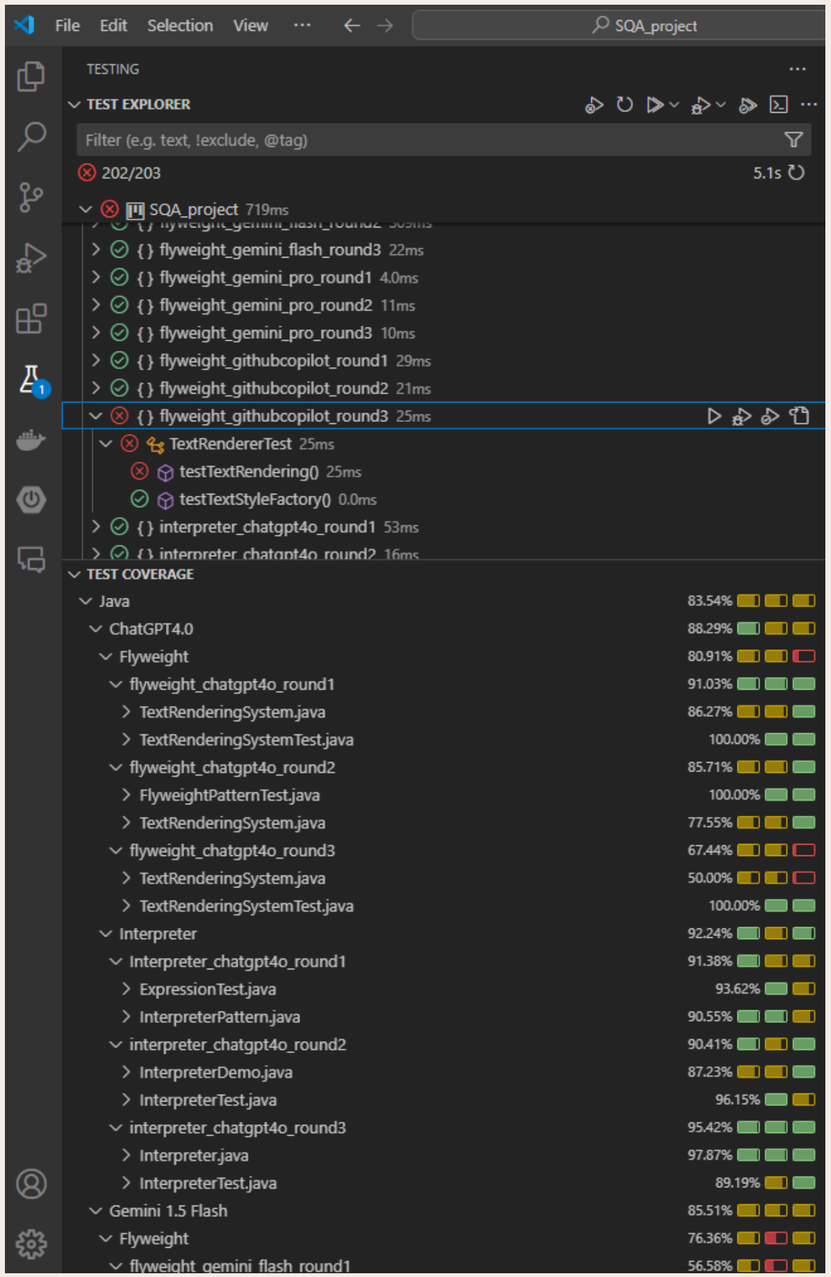
ตัวอย่างผลการทดสอบโดยใช้ Pytest ของ Python

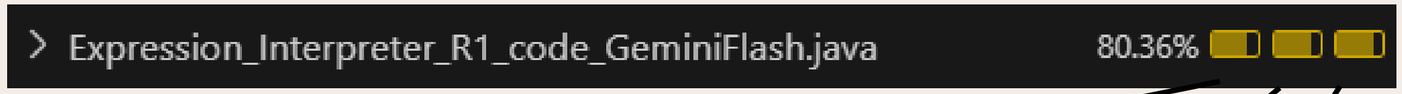
cov-branch ของ pytest เป็นตัวเลือกที่ใช้ในการตรวจสอบการครอบคลุมของโค้ด (code coverage)



1. Stmts (Statements)
หมายถึง: จำนวนของคำสั่งทั้งหมดในโค้ดที่สามารถดำเนินการได้
ตัวอย่าง: ถ้าโค้ดมี 28 บรรทัดที่สามารถรันได้ จะนับเป็น 28 statements
2. Miss (Missed Statements)
หมายถึง: จำนวนของคำสั่งที่ไม่ได้ถูกทดสอบหรือไม่ได้ถูกเรียกใช้ในการทดสอบ
ตัวอย่าง: ถ้าใน 28 statements มี 0 statements ที่ไม่ได้ถูกเรียกใช้ จะบอกว่า Miss = 0
3. Cover (Coverage)
หมายถึง: อัตราส่วนของคำสั่งที่ถูกทดสอบ (covered statements) เทียบกับจำนวนคำสั่งทั้งหมด
ตัวอย่าง: ถ้ามี 28 statements และ 0 missed statements, coverage จะเป็น 100%
4. Branch (Branches)
หมายถึง: จำนวนของ branches (สาขา) ในโค้ด เช่น เงื่อนไขที่เกิดจาก if, for, while เป็นต้น
ตัวอย่าง: ถ้าโค้ดมี 4 branches จะนับเป็น 4 branches
5. BrMiss (Branch Missed)
หมายถึง: จำนวน branches ที่ไม่ได้ถูกทดสอบในระหว่างการทดสอบ
ตัวอย่าง: ถ้ามี 4 branches และมี 0 branches ที่ไม่ได้ถูกทดสอบ จะบอกว่า BrMiss = 0

ตัวอย่างผลการทดสอบโดยใช้ JUnit ของ Java





1. Statements covered (%)
หมายถึง: เปอร์เซนต์จำนวนของคำสั่งทั้งหมดในโค้ดที่ถูกทดสอบ
จากตัวอย่างที่ยกมาได้ 78.72 %

2. Functions covered (%)
หมายถึง: เปอร์เซนต์จำนวนของฟังก์ชันทั้งหมดในโค้ดที่ถูกทดสอบ
จากตัวอย่างที่ยกมาได้ 86.67 %

3. Branches covered (%)
หมายถึง: วัดว่ามีการทดสอบทุกสาขา (branch) ในคำสั่งเงื่อนไข
หรือไม่ จากตัวอย่างที่ยกมาได้ 81.36%

มาตรวัดคุณภาพของโค้ด (Code metric)

Cyclomatic Complexity

มาตรวัดที่ใช้วัดความซับซ้อนของโค้ดโดยการนับจำนวนเส้นทางการไหล (control flow paths) โดยมาตรวัดนี้จะคำนวณจากจุดตัดสินใจในโค้ด เช่น คำสั่งเงื่อนไข (conditional statements) และลูป (loops)

Cyclomatic complexity = E - N + 2*P
โดยที่,

- E = จำนวนของ edges (เส้นเชื่อม) ในกราฟ
- N = จำนวนของ nodes (จุด) ในกราฟ
- P = จำนวนของ nodes ที่มีจุดออก (exit points)

Cyclomatic Complexity	Code status	Testability	Maintenance costs
1~10	Clear	High	Low
10~20	Complex	Medium	Medium
20~30	Very complex	Low	High
>30	Unreadable	Unmeasurable	Very high

ที่มา : https://www.alibabacloud.com/blog/clean-code---be-a-thinking-programmer-instead-of-a-code-farmer

ตัวอย่างการวัดผลของโค้ดโดยใช้ Codalyze

AI: Chatgpt
Design pattern: Flyweight (Python)
รอบที่ 1

Function Name	Start Line	End Line	Cyclomatic Complexity (Threshold: 10)	Lines of Code (Threshold: 50)	Parameter Count (Threshold: 4)
__init__	5	9	1	5	⚠ 5
render	11	25	1	14	⚠ 6
get_font	32	36	2	5	⚠ 5
__init__	40	41	1	2	1
render_text	43	45	1	3	⚠ 10

1. Function Name: ชื่อของฟังก์ชัน
2. Start Line: บรรทัดที่ฟังก์ชันเริ่มต้น
3. End Line: บรรทัดที่ฟังก์ชันสิ้นสุด
4. Cyclomatic Complexity: ความซับซ้อนของโค้ดในฟังก์ชันนั้นๆ (ค่าที่สูงขึ้นหมายถึงโค้ดที่ซับซ้อนมากขึ้น)
5. Lines of Code (Total S): จำนวนบรรทัดของโค้ดทั้งหมดในฟังก์ชัน
6. Parameter Count: จำนวนพารามิเตอร์ที่ฟังก์ชันรับเข้า

สรุปผลการทดสอบ

จากผลการทดลอง สามารถสรุปได้ดังนี้:

1. ChatGPT-4:
   - มีความสามารถในการสร้างโค้ดที่มี Cyclomatic Complexity ต่ำ ซึ่งหมายถึงโค้ดมีความซับซ้อนน้อยและง่ายต่อการบำรุงรักษา
   - อย่างไรก็ตาม พบปัญหาเรื่อง ความไม่สม่ำเสมอ ในการสร้างโค้ด แม้จะใช้ Prompt ที่คล้ายกัน ทำให้โค้ดบางส่วนไม่ตรงกับมาตรฐานที่คาดหวัง
2. GitHub Copilot:
   - โดดเด่นในเรื่องของการสร้างทั้ง โค้ด และ Test Code ที่มีคุณภาพสูง
   - สามารถทำให้ได้ทั้ง Statement Coverage และ Branch Coverage สูงที่สุด
   - มีความสามารถในการสร้างโค้ดที่ตรงตาม Requirements ในบางครั้ง
3. Gemini 1.5 Pro และ Gemini 1.5 Flash:
   - สร้างโค้ดที่มีคุณภาพ แต่ไม่ถึงขั้นสูงสุดเหมือนกับ GitHub Copilot
   - พบว่ามี ความซับซ้อน ในการสร้างโค้ดที่สูงกว่าในบางกรณี

สมาชิกกลุ่ม

นายชนินทร์ ธัญสิริพัฒนธาดา รหัสนักศึกษา 653380125-2
นายชลพัฒน์ ปิ่นมุนี รหัสนักศึกษา 653380126-0
นายปวีณวัฒน์ สุขร่วม รหัสนักศึกษา 653380136-7

Thank You