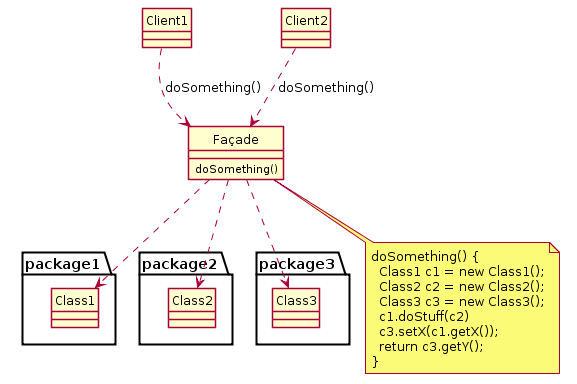
**Facade Pattern** เป็นเทคนิคการออกแบบที่่มีวัตถุประสงค์เพื่อลดความ Complex ของการเขียน Program ลง โดยการนำ Code ที่การเรียกใช้งานอาจประกอบด้วยหลายขั้นตอน มา Wrap ไว้ใน Object ใหม่ ที่่อาจเรียกใช้งานได้ง่ายขึ้น โดยจากการเรียกใช้งานหลายขั้นตอน พอ Wrap แล้วอาจเหลือเพียงขั้นตอนเดียว เป็นต้น

จากในภาพ Client1 และ Client2 เรียกใช้งาน doSomething() ผ่าน Facade Object แทนที่จะต้องสร้าง Class ขึ้นมาทีละ Class และเรียกใช้งานเอง ซึ่งค่อนข้างจะวุ่นวาย สังเกตุว่าการใช้ Facade Pattern จะช่วยทำให้ Code ที่เราเขียนสั้นลง เข้าใจได้ง่ายขึ้น อีกทั้งยังดูแลได้ง่ายขึ้นอีกด้วย

Facadeเป็นobject ที่ห่อหุ้มการทำงานที่ซับซ้อนเอาไว้แล้วแสดง interface ง่ายๆให้client เอาไปใช้

Facade design pattern นั้นถูกใช้บ่อยในระบบขนาดใหญ่ เพราะมีคลาสจำนวนมหาศาล และบางคลาสก็ไม่มี source code อีก แพทเทิร์นนี้จะเข้าไปช่วยซ่อนความซับซ้อนเหล่านั้นเอาไว้

**ประโยชน์ของ Facade pattern**

-ทำให้ software library นั้นๆ ง่ายต่อการใช้งาน ง่ายต่อการเข้าใจ ง่ายต่อการทดสอบ

-ทำให้ software library นั้นอ่านง่าย

-ลดจำนวน dependencies ของ client ลง เพราะตัว library เองซ่อนความซับซ้อนเหล่านั้นไว้

-ซ่อน APIs ที่เป็ร poorly designed โดยการครอบด้วย API ใหม่ที่สวยงาม

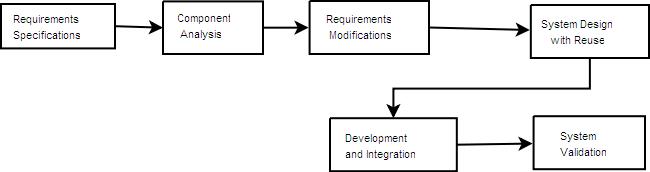
จาก diagram จะเห็นว่าคลาสที่เป็น facade นั้นจะมีเมธอด doSomething() ไว้ให้ client นำไปใช้ แต่เจ้า doSomething() เนี่ย ไม่ได้ถูก implement การทำงานในคลาสนี้ทั้งหมด แต่เป็นการเรียกใช้ความสามารถของคลาสอื่นๆ แทน

ดูๆ เหมือนจะคล้ายกลับ Adapter และ Decorator แต่สิ่งที่แพทเทิร์นนี้ต้องการสื่อจริงๆ คือซ่อนการทำงานที่ซับซ้อนเอาไว้ ไม่ใช่แค่ทำให้เข้ากันได้กับ Client หรือเพิ่ม Behavior แบบ Decorator

**ตัวอย่างการใช้งาน** วัตถุบนโลกจริงเราส่วนใหญ่ก็เป็น facade ทั้งนั้น เพราะมันปิดการทำงานที่ซับซ้อน แล้วแสดงให้เราเห็นแค่ interface ง่ายๆ เช่น ทีวีก็ประกอบด้วยชิ้นส่วนต่างๆ มากมาย แต่เรารู้แค่ว่าถ้าจะเปิดกดตรงนี้นะ ถ้าจะปรับเสียง เปลี่ยนช่อง ก็กดตรงนู้น อะไรแบบนั้น

จากตัวอย่างจะเห็นว่าเรา (คลาส You) เวลาจะเปิดใช้เครื่อง computer ก็แค่สั่ง start() ซึ่งจริงๆ แล้วการทำงานมันมีมากกว่านั้น แต่ ComputerFacade ซ่อนเอาไว้ไม่ให้เห็น ตรงนี้แหละครับที่บอกว่า facade ทำให้ client มี dependency ลดลงมาก ใช้งานง่ายขึ้นมาก อ่านโค้ดง่ายขึ้นมาก เพราะ client จะรู้เฉพาะสิ่งที่ควรรู้เท่านั้น (หลักการเดียวกับ abstraction)

**3. CBSE (Component-Based Software Engineering) (การนำ Component ที่มีอยู่แล้วกลับมาใช้ใหม่)**



ข้อดี

1. รวดเร็วขึ้น

2. ประหยัดต้นทุน

3. ระบบมีความเชื่อมั่นเพิ่มมากขึ้น

4. ความเสี่ยงที่จะเกิดความล้มเหลวลดลง

ข้อเสีย

1. หา Component ที่ตรงตามความต้องการได้ยาก

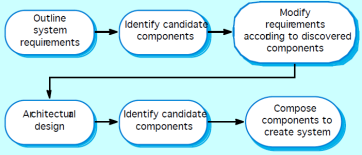
2. ยากในการปรับปรุง Component ให้ตรงตามความต้องการ

**Integration Test** คือ การนำส่วนย่อยแต่ละส่วนมาทดสอบร่วมกันกับส่วนย่อยส่วนอื่นๆดู เพื่อทดสอบการทำงานร่วมกันระว่างส่วนย่อยแต่ละส่วนโดยส่วนย่อยนั้นอาจจะยังไม่เสร็จ ยังไม่มีจริง ซึ่งเราสามารถเขียนขึ้นมาทดสอบชั่วคราวก่อนดูว่ามันสามารถทำงานร่วมกันได้จริงไหม และถ้ารวมส่วนย่อยหลายๆส่วนเข้าด้วยกันก็เกิดเป็น System ขึ้นมา

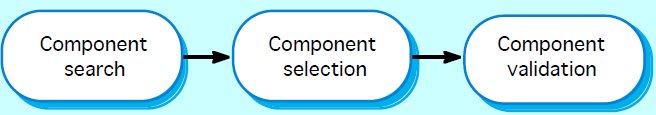
**Integration testing :** เป็นทดสอบการทำงานฟังก์ชันต่างๆ ที่สัมพันธ์กันภายในระบบย่อยของโปรแกรม ว่ามีการทำงานได้อย่างมีประสิทธิภาพหรือไม่ โดยการทดสอบนี้จะต้องอ้างอิงถึงขั้นตอนการออกแบบระบบหรือโปรแกรม เพื่อให้รู้ว่าภายในระบบหรือโปรแกรมนั้นมีระบบย่อยๆอะไรบ้าง

**System Integration Test** คือ ทดสอบการทำงานร่วมกันทั้งระบบว่าแต่ละส่วนนั้นสามารถทำงานร่วมกันได้ครบทุกส่วนและถูกต้องหรือไม่ จากนั้นก็ไปทำขั้นตอนต่อไปคือ system test

**THE CBSE PROCESS THE**

****

**COMPONENT IDENTIFICATION PROCESS**

****

**TYPES OF COMPOSITION**

**● Sequential composition องค์ประกอบลำดับ ที่เป็นส่วนประกอบประกอบด้วยดำเนินการในลำดับ นี้เกี่ยวข้องกับการเขียนให้อินเตอร์เฟซของแต่ละองค์ประกอบ ● Hierarchical composition องค์ประกอบตามลำดับชั้น องค์ประกอบหนึ่งที่เรียกร้องให้บริการของผู้อื่น ให้อินเตอร์เฟซขององค์ประกอบหนึ่งคือประกอบกับต้องมีอินเตอร์เฟซของผู้อื่น ● Additive composition องค์ประกอบเสริม อินเตอร์เฟซที่สองส่วนประกอบจะใส่กันเพื่อสร้างองค์ประกอบใหม่**

**Advantages of COTS**

ค่าใช้จ่ายในการพัฒนาจะลดลง , เวลาในการพัฒนาจะลดลง , ระบบที่ซับซ้อนสามารถสร้างขึ้นโดยการนำที่มีอยู่ก่อนส่วนประกอบ , การทดสอบความพยายามที่จะลดลง

**What is Software Integration Testing?**

กิจกรรมการทดสอบที่รวมองค์ประกอบซอฟต์แวร์กันในรูปแบบระบบที่สมบูรณ์เพื่อดำเนินการรวมซอฟต์แวร์ที่มีประสิทธิภาพการทดสอบบูรณาการกลยุทธ์การทดสอบชุดบูรณาการที่มีความจำเป็น

**System Integration Techniques**

**Common approaches to perform system integration testing**

**• Incremental**

**• Big-bang**

**• Top-down**

**• Bottom-up**

**• Sandwich**

**Major testing focuses:**

• Interfaces between modules

(or components)

• Integrated functional features

• Interacting protocols and messages

• System architectures

**Integration Testing**

■ถือว่าหน่วยได้รับการทดสอบเป็นรายบุคคล

■หน่วยการทดสอบการทำงานร่วมกัน

■ระบุข้อผิดพลาดในการเชื่อมต่อระหว่างหน่วย

■แนวทาง • Big bang • Top-down • Bottom-up • Sandwich