Aspect-Oriented Programming

การพัฒนาซอฟต์แวร์ได้มีการพัฒนามานานแล้ว ประโยชน์อันหนึ่งของการพัฒนาซอฟต์ที่ได้เกิดขึ้นจริง โดยการผูกมัดติดกับการค้นหาเทคนิคโมเดลปัญหาที่มีประสิทธิภาพ 2-3ปีที่ผ่านมา Methodology สำหรับการแก้ปัญหาได้แบ่งแยกย่อยให้เป็น functions ซึ่งจะประกอบไปด้วยจำนวนโค้ดหลายบรรทัด Methodologyทำงานได้ดีแต่มันได้รับจากสถานะเริ่มต้นของระบบ ผ่านจำนวนตัวแปรแบบ Global ที่หลากหลาย ซึ่งสามารถถูกแก้ไขโดยโค้ดโปรแกรมใน Application การกำเนิดของ Object-oriented Methodologies ได้ดึงสถานะของระบบไปใน Object เดียวๆ ที่ซึ่งมันสามารถถูกสร้างการป้องกันตัวแปรและควบคุมโดยผ่าน Method และ Logic

ปัจจุบันนี้นักพัฒนายังมีความยากที่จะอธิบายปัญหาในลักษณะ Modular ที่สมบูรณ์และแบบจำลอง Encapsulate ถึงแม้ว่าการแก้ปัญหาไปสู่ Object ตามความรู้สึกแล้ว บางหน้าที่ (Functionality)ต้องถูกสร้างสามารถ Across Object อย่างเหมาะสม Aspect-oriented programming (AOP) เป็นหนึ่งในการแก้ปัญหาของหารสร้างอย่างดี

Where OOP has brought us

การวิเคราะห์แบบ Object-oriented และการโปรแกรม (OOADP) ได้ถูกพิสูจน์จนสำเร็จในทั้งโปรเจคใหญ่หรือเล็ก ซึ่งเป็นการยกระดับของการพัฒนาซอฟต์แวร์ให้ดีขึ้น โดยมาแทนที่ การเขียนโปรแกรมแบบเดิม (Functional-decomposition) โประโยชน์ของ OOADP มีดังนี้

- สามารถเขียนโปรแกรมแล้วนำ Components กลับมาใช้ได้ใหม่ได้
- แยกเป็น Modularity ได้
- มีความซับซ้อนในการเขียนโปรแกรมน้อย
- ลดต้นทุนในการดูแลรักษาซอฟต์แวร์

ประโยชน์แต่ละข้อที่กล่าวมาแล้วจะเปลี่ยนแปลงไปตามความสำคัญของนักพัฒนา เช่น

Modularity คือการครอบคลุมโครงสร้างโปรแกรม ซึ่งทำให้เข้าใจซอฟต์แวร์ได้ง่าย

What OOADP did for computer science

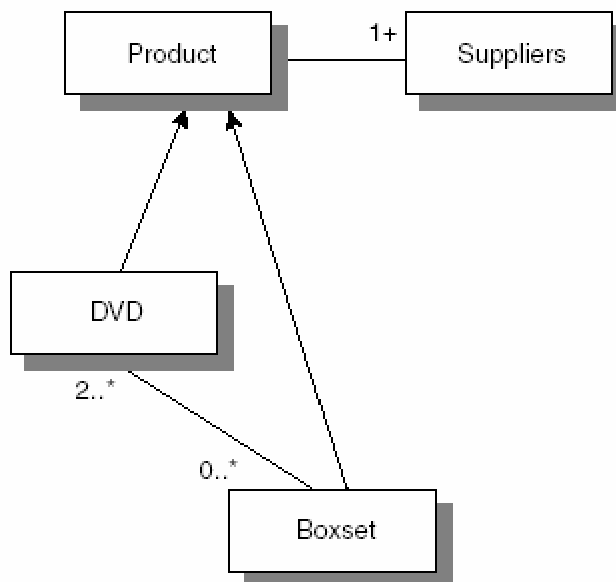
Methodology แบบ Object-oriented (ประกอบไปด้วย การวิเคราะห์ การออกแบบและการเขียนโปรแกรม) ทำให้วิทยาศาสตร์คอมพิวเตอร์มีความสามารถจำลองหรือออกแบบซอฟต์แวร์มากกว่าการออกแบบระบบในโลกความเป็นจริง เครื่องมือหลักที่ใช้ในการจำลอง (modeling) คือ **Object** มันเป็นตัวแทนของ Component หลักของปัญหาหลัก ซึ่งมี **Attributes** ไว้อธิบายสถานะของ Object และพฤติกรรมของ Object เป็นการอธิบายการทำงานของ Object ในการเปลี่ยนสถานะ ดังตัวอย่าง ถ้าเราจะออกแบบระบบจัดการขาย DVD การออกแบบแบบ OO อาจจะประกอบไปด้วย Object ดังนี้ Product, DVD และ Boxset อาจจะมีมากกว่านี้ก็ได้

By Mr.Surachai Samattasribut
Computer Engineering School
Suranaree Univesity of Technology

Object แต่ละ Object จะต้องมี Attributes และ พฤติกรรมของมันเอง Product อาจจะออกแบบดังนี้

- Attributes
  - Price
  - Title
  - Suppliers
- Behaviors
  - Assign price
  - Assign title
  - Get suppliers

เนื่องจาก Class Product ของระบบต้องการประกอบด้วย Attributes และพฤติกรรม (Behavior) มากกว่านี้ แต่การเพิ่ม Objects ของการผลิตจะทำให้เหมาะสมกับจุดประสงค์ของรายงานนี้ ในการออกแบบจะต้องสร้างความสัมพันธ์ระหว่าง Production และ Supplier หลังจากนั้นก็จะได้ DVD Objects และ Boxset Objects ออกมาดังรูปด้านล่างนี้



ตัวอย่างแบบจำลอง Class

จุดประสงค์หนึ่งในการออกแบบ Object คือการรวมข้อมูล (Data) และวิธีการ (Method) ที่เกี่ยวข้องกันเข้าด้วยกันเรียกว่า **Encapsulation** เพื่อการจัดการข้อมูลหรือเป็นการแก้ไขเปลี่ยนแปลงสถานะ (State) ภายใน Objects นั้นๆ ซึ่งไม่ควรมี Functions ภายนอก Object สามารถเข้ามาแก้ไขข้อมูลภายใน Product Object ได้โดยตรง และ Product Object ก็ไม่สามารถไปแก้ไขข้อมูลของ Objects อื่นได้โดยตรงเช่นกัน แทนที่ Suppliers Object อาจจะส่ง Message ไปให้ Product Object ที่มีการเรียก Class Supplier โดยการสร้าง Supplier Object ภายใน Class Product เมื่อ Message ถูกส่งจาก Object หนึ่งไปยัง Objects อื่นๆ Object ที่สร้าง Object นั้นไว้ภายในก็สามารถเรียกใช้ Methods ที่เป็นชนิด Public ได้ ดังนั้น Methods ที่ไม่ต้องการให้ Objects อื่นเรียกใช้ได้ก็ควรให้เป็นชนิด Private

การออกแบบระบบ OO ในลักษณะ Functions ของระบบ, การแก้ไข (Debugging) เมื่อมีปัญหา และการขยายระบบ Object ทั้งหมดในระบบรู้บทบาทและการแสดงออกของตัวมันเอง ซึ่งเมื่อดูจากมุมมองแบบง่ายๆระบบเป็นแค่เพียงการรวมกลุ่มของ Objects เท่านั้น ซึ่งเมื่อระบบทำงานและส่ง Message ไปยังแต่ละ Object เมื่อมีการเรียกใช้ Object นั้นเพื่อจะดูข้อมูล (Information และเปลี่ยนสถานะใน Objects อื่น

เมื่อเทคโนโลยี OO ได้ขยายกว้างออกไป ก็ได้มีคำศัพท์ใหม่ๆเกิดขึ้น ดังที่คุณรู้ว่า หนึ่ง Object คือการประกาศใช้ Class หนึ่ง Class ซึ่ง Class ก็คือ Abstract Data-Type ที่ใช้ในการจำลอง Object ในระบบ หนึ่ง Class จะถูกสร้างตามความต้องการ (Requirement) ที่ได้มาจาก Analysis Phases (เฟสการวิเคราะห์ระบบ) แต่บางครั้ง Class ก็จะถูกสร้างใน Construction Phase (เฟสการสร้าง Code Program) ตามความต้องการที่ได้ Comments มากับ Class ความต้องการ (Requirement) เหล่านี้และ Class สามารถถูกเชื่อมโยงเข้าด้วยกันได้โดย Concern

Concern ก็คือ Functionality บางอย่าง หรือความต้องการที่จำเป็น (Non-Functionality) ในระบบ ซึ่งได้มีการสร้างไว้ในโครงสร้างของ Code Program การนิยามนี้ยอมให้ Concern ไม่มีขีดจำกัดในระบบ OO (โครงสร้างในระบบสามารถมี Concern) ในระบบทั่วไปแล้วจำนวนมากๆ ของ Concern ต้องการถูกเตรียมในแบบแผนสำหรับระบบที่ทำให้สมบูรณ์ตามวัตถุประสงค์ คนที่ออกแบบระบบจะต้องเจอกับการสร้างระบบ ซึ่งใช้ Concern แต่ต้องไปผิดกับกฎของ Methodologies ที่กำลังถูกใช้ เมื่อ Concern ถูกสร้างให้ได้ครบทั้งหมดตามที่ต้องการของระบบ และมีการทดสอบแล้ว ระบบก็เสร็จสมบูรณ์

Problems Resulting from OOP

หนังสือหรือบทความส่วนใหญ่ที่เกี่ยวข้องกับ OOP จะพูดเสมอว่า OOP ยอมสำหฉับการ Encapsulation ของข้อมูลและ Methods ที่เกี่ยวข้องกับ Object นั้นๆ ในอีกความหมายหนึ่ง Object ควรเป็นหน่วยที่บรรจุตัว Object เองกับการไม่รู้เกี่ยวกับสิ่งแวดล้อมที่ Object อื่น ไม่อนุญาตให้รู้ได้ Class เป็นตัว Cutter สำหรับ Objects ในระบบ และมันสร้าง Concern สำหรับระบบ จุดประสงค์ของ Class คือการ Encapsulation ตัว Code Program ที่ต้องการสำหรับ Concern โชคไม่ดีนัก วิธีการนี้เป็นไปได้แต่ไม่เสมอไป ลองพิจารณา 2Concerns ต่อไปนี้

Concern 1: ระบบจะเก็บราคาขายส่งของผลิตภัณฑ์ (DVD) ทั้งหมด

Concern 2: เมื่อมีการเปลี่ยนแปลงราคา จะต้องถูกบันทึกไว้สำหรับเป็นข้อมูล

ในโลกของ OO Concern แรกจะสร้าง Product Class เป็น Abstract Class เพื่อจัดการกับ Functionality ธรรมดาของผลิตภัณฑ์ทั้งหมดในระบบ

```
public abstract class Product {

	private real price;

	Product() {
		price = 0.0;
	}
	public void putPrice(real p) {
		price = p;
	}
	public int getPrice() {
		return price;
	}
}
```

จาก Class Product ถูกสร้างขึ้นตามความพอใจความต้องการใน Concern 1 หลักการของ OO ก็สามารดูแลรักษาได้ เพราะว่าการ Encapsulates สิ่งที่จะเป็นเพื่อเก็บข้อมูลของราคา (Price) เมื่อมี Functionality ที่เหมือนกันก็สามารถถูกสร้างได้ง่ายๆ ในโครงสร้างที่ใช้ตัวแปรแบบ Glocal Array

ตอนนี้ก็มาดู Concern ที่ 2 บ้าง ซึ่งต้องที่จะบันทึก (Logging) ราคาที่มีการเปลี่ยนแปลงทั้งหมดไม่ว่าจะเปลี่ยนแปลงราคาด้วย Operations (= Methods) ใดก็ตาม ในตัวของ Concern เองก็ไม่ได้ขัดแย้งกับ Concern แรกและสามารถสร้างได้อย่างง่าย ตามดังต่อไปนี้

```
public class Logger {

	private OutputStream ostream;

	Logger() {
		//open log file
	}
	void writeLog(String value) {
		//write value to log file
	}
}
```

ในการเรียกใช้ Logger ก็เพิ่ม Method writeLog() ไปใน Code Program ที่ต้องต้องให้จะให้บันทึกการเปลี่ยนแปลงราคาขายส่งของผลิตภัณฑ์ ซึ่งในที่นี้มีเพียง Product Class เดียวที่ควรมีการบันทึกการเปลี่ยนแปลงราคา จะได้ Code ของ Product ใหม่ดังนี้

```
public abstract class Product {

	private real price;
	Logger loggerObject;

	Product() {
		price = 0.0;
		loggerObject = new Logger();
	}
	public void putPrice(real p) {
		loggerObject.writeLog("Changed Price from" + price + " to " + p);
		price = p;
	}
	public int getPrice() {
		return price;
	}
}
```

เมื่อลองพิจารณาแนวความคิดของ Encapsulation และ Modularity ใน OO Methodologies ในการเพิ่ม Code ใน Product Class เพื่อจัดการกับ Concern ที่ 2 ในระบบนั้นมันก็จะแสดงให้เห็นว่าไม่ได้เป็นไปตามแนวความคิดของ Encapsulation แล้ว ซึ่ง Class ไม่ได้จัดการเพียงแค่ Concern เดียวเท่านั้น แต่มันต้องเป็นที่เติมเต็มความต้องการของ Concern อื่นด้วย Class ก็เลยเป็นการ Crosscut โดย Concerns ในระบบ

Crosscutting เป็นตัวแทนเหตุการณ์เมื่อความต้องการของระบบถูกพบ โดยการเพิ่ม Code Program ใน Objects ผ่านไปยังระบบ แต่ Code Program ไม่มีความสัมพันธ์เกี่ยวข้องกับ Functionality ของ Object นั้นเลย อย่างเช่น Class Product ที่มีการ Logging อยู่ด้วย

พิจารณาว่าอะไรจะเกิดขึ้นกับ Class Product ถ้ามีการเพิ่ม ข้อมูลของเวลา, ผู้ที่เข้ามาเปลี่ยนแปลงราคาและเกี่ยวกับฐานข้อมูลเข้าไป Concern เหล่านี้ถูกออกแบบใน Object Product ใช่หรือไหม? โครงสร้างและภาษา OO ก็ไม่มีทางเลือกเมื่อมีการ Crosscutting Concern ซึ่ง Concern จะถูกบังคับให้เป็นส่วนหนึ่งของ Concern อื่น ดังนั้นก็เท่ากับว่าเป็นการแหกกฎของ Methodologies ที่มีอยู่มากมาย

การผสมของ Concern หลายอย่าง ก่อให้เกิดเงื่อนไขที่ถูกเรียกว่า Code Scattering (Code ที่กระจัดกระจาย) และ Tangling (Code ที่ยุ่งเหยิง) Code ที่กระจัดกระจายเป็น Code ที่จำเป็นเพื่อเติมเต็ม Concern ที่ถูกหระจายไปทั่ว Class ที่ถูกต้องการเติมเต็ม Concern อื่น Code ที่ยุ่งเหยิงมักจะเป็นการเรียกใช้ Method หนึ่งหรือ Class เพื่อสร้าง Concern หลายอย่าง ทั้งสองปัญหานี้ได้ทำลายหลักการพื้นฐานของ OO และเป็นสาเหตุทำให้นักออกแบบปวดหัวไปตามๆกัน

ลองดูว่า Class Product ต่อไปนี้ ที่อยู่ในรูปของ Pseudocode ด้วยบางส่วน การเพิ่ม Functionality นี้จำเป็น แต่มันไม่ควรจะอยู่ส่วนของ Class Product

```
public abstract class Product {

        private real price;
        Logger loggerObject;

        Product() {
                price = 0.0;
                loggerObject = new Logger();
        }
        public void putPrice(real p) {
                //start timing
                //Check user authentication
                loggerObject.writeLog("Changed Price from" + price + " to "+ p);
                price = p;
                // log if problem with authentication
                //end timing
                //log timing
        }
        public int getPrice() {
                //check user authentication
                return price;
        }
        public void persistIt() {
                //start timing
                //save this object
                //end timing
                //log timing
        }
}
```

Class Product ได้ถูกสร้างขึ้น Concrete Class DVD ถูกกำหนดขึ้น Class DVD สืบทอด Functions ทั้งหมดใน Class Product และเพิ่ม Attributes 2-3 ตัว Class DVD ประกอบไปด้วย Attribute และ Methods ที่เกี่ยวข้องสำหรับการนับจำนวนที่ขาย DVD ได้ ข้อมูลที่สำคัญนี้ซึ่งควรมีการ Logging ทุกขั้นตอน

```
public class DVD extends Product {

	private String title;
	private int count;
	private String location;

	public DVD(String inTitle) {
		super();
		title = inTitle;
	}
	private void setCount(int inCount) {
		//start timing
		//check user authentication
		count = inCount;
		//end timing
		//log timing
	}
	private int getCount() {
		return count;
	}
	private void setLocation(String inLocation, int two) {
		//start timing
		//check user authentication
		location = inLocation;
		//end timing
		//log timing
	}
	private String getLocation() {
		return location;
	}
	public void setStats(String inLocation, int inCount) {
		//start timing
		//check user authentication
		setLocation(inLocation, 0);
		setCount(inCount);
		//end timing
		//log timing
	}
	}
```

จะสังเกตุเห็นปัญหาของ Code ไหม? การ Logging ไม่ได้ถูกประกอบไปใน Methods ซึ่งเปลี่ยนข้อมูลการนับ นักพัฒนาซอฟต์แวร์ลืม Concern นี้เมื่อสร้าง Class ใหม่

By Mr.Surachai Samattasribut
Computer Engineering School
Suranaree Univesity of Technology

Results of tangled code

ผู้พัฒนาไม่ได้อยู่ในอุตสาหกรรมยาวนานพอที่จะค้นพบผลกระทบของการยุ่งเหยิงและการกระจัดกระจายของ Code ซึ่งผลกระทบบางอย่างมีดังนี้

- การเปลี่ยนแปลง Class ทำได้ยาก
- ไม่สามารถนำ Code กลับมาใช้ใหม่
- ไม่สามารถอ่าน Code ได้เข้าใจ

นักวิศวกรรมและผู้จัดการเป็นผู้ซึ่งต้องการที่จะ Refactor (แก้ไข Code ของเก่าโดยแก้ไขทั้งโปรแกรม) Code ที่อ่านไม่รู้เรื่อง ถ้า Code ที่เขียนมาอย่างดี อ่านแล้วเข้าใจง่าย มีการใช้ Object รูปแบบที่ดี (Well-defined) ก็สามารถเทียบเคียงอัตราต้นทุนและกำไรอย่างเห็นได้ชัดเจน ถ้าเวลาและเงินสามารถถูกนำมาเป็นเหตุผล Component ในระบบก็สามารถถูกนำมาใช้ใหม่โดยการ Refactor อย่างไรก็ตามส่วนมาก Code ของ Components ถูกนำมาใช้ใหม่ และประกอบเข้ากันใหม่จนราคาต้นทุนสูงเกินไปตามความหมายแบบเดิม อย่างไรก็ตาม **AOP** ยอมให้การ Refactor ถูกแสดงบนระดับที่ต่างกัน และในลักษณะที่ช่วยกำจัด Code ที่ยุ่งเหยิงบางส่วน

โปรเจค Jakarta Tomcat เป็นตัวอย่างที่กำหนดให้ Code ซึ่งมีการใช้ Logging อยู่ใน Code ผลลัพธ์ของโปรเจคที่แสดงให้เห็นว่า Code ส่วนการ Logging ไม่ได้มีเพียงหนึ่งที่เท่านั้น และไม่มีโอกาสที่จะแยกออกจาก Code ได้

เมื่อนักวิเคราะห์ของ Tomcat แสดงให้เห็นว่า Code Tangling เป็นปัญหาหลักปัญหาหนึ่ง เพียงแค่คิดก็ฝันร้ายแล้ว ถ้าต้องการเปลี่ยน Code ของการ Logging การที่ Code ดูยุ่งเหยิงก็สามารถทำให้ Applications เสร็จสมบูรณ์ กำรหนดบาง Functionality เช่น Logging Code เป็นผลที่ได้มาจากความต้องการใน Code ที่ยุ่งเหยิงแล้ว เราจะเรียกว่ามัน Crosscut ระบบ Crosscutting ไม่ใช่ความต้องการหลักของระบบเสมอไป เพียงแค่การ Logging ไม่ถูกต้องการสำหรับ Software Application เพื่อคุณสมบัติ Function แต่บางครั้งมันถูกต้องการในกรณีของการบันทึกว่ามีผู้ใช้อะไรบ้าง

How AOP solves OOP problems

Aspect-oriented programming (AOP) เป็นต้นแบบที่ถูกสร้างเพื่อแก้ไขปัญหาที่ได้กล่าวมาแล้ว โดยเป็นวิธีที่ไม่ยากและไม่ซับซ้อนด้วย Subject-oriented programming (SOP) และ Multidimensional separation of concerns (MDSOC) AOP ไม่ใช่แนวความคิดใหม่ พื้นฐานของมันจริงๆเป็นการแยกของขบวนการ Concerns แต่มันก็ถูกนำมาใช่ซึ่งผ่านการทำงานโดย Gregor Kiczales และวิทยาลัยของเขาเองที่ Xerox's PARC (www.parc.com/groups/csl/projects/aspectj/)

การเรียนรู้การเขียน AOP เพื่อแก้ปัญหา Crosscutting ของ Concerns เพียงแค่เรียนรู้คำศัพท์หลักเพียงเล็กน้อยของ AOP มันสนับสนุนการนำกลับมาใช้ใหม่และ Modularity ของ

Code และกำจัด Code ที่กระจัดกระจายหรือไม่เป็นระเบียงยุ่งเหยิง ด้วยภาษา Java และ AspectJ (หรือ JBoss AOP) ซึ่ง AOP กำลังเป็นสิ่งที่ใหญ่ในวิทยาศาสตร์คอมพิวเตอร์ ตั้งอต่มี OOP มา

What is AOP?

AOP คือ ต้นแบบซึ่งสนับสนุนจุดประสงค์พื้นฐาน 2 จุดประสงค์

1. ยอมให้สามารถแยก Concerns ได้
2. จัดการกลไกสำหรับลักษณะ Concerns ซึ่งเกิดการ Crosscut กับ Components อื่น

AOP ไม่ใช่สิ่งที่จะมาแทนที่ OOP หรือ Object-based methodologies อื่นๆ มันแค่สนับสนุนดังต่อไปนี้

1. การแยกของ Components
2. การใช้ Class ทั่วไป
3. แยก Aspect จาก Components

ในตัวอย่างที่ผ่านมา AOP ถูกออกแบบให้สนับสนุนการแยกของตัวอย่าง Concerns และมี Class Logger และ Class Product มันยังจัดการ Crosscutting ด้วย ซึ่งเกิดเมื่อ Logging ถูกเรียกใช้ใน Components อื่น

Development Process with AOP

การที่เข้าใจแนวความคิดของ AOP ช่วยการเกิด Crosscutting ได้อย่างไร ลองดูตัวอย่าง Concerns ใหม่

Concern 1: ระบบจะเก็บราคาขายส่งของผลิตภัณฑ์ (DVD) ทั้งหมด

Concern 2: เมื่อมีการเปลี่ยนแปลงราคา จะต้องถูกบันทึกไว้สำหรับเป็นข้อมูล

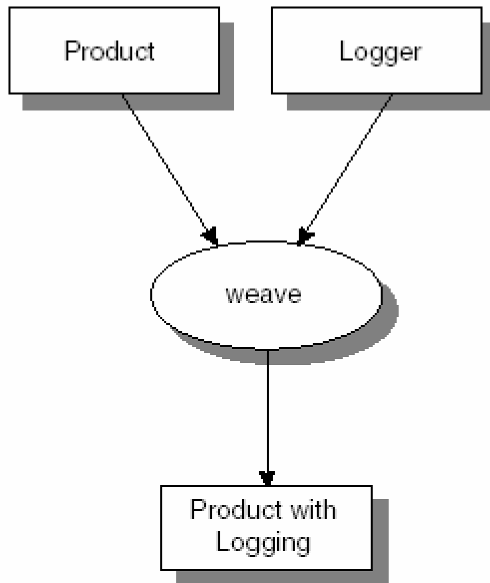
จาก 2Classes ที่ได้สร้างไว้เพื่อแยก 2Concerns นี้ อย่างไรก็ตาม Concern 2 ถูกสร้างขึ้น มันก็จะถูกเรียกจาก Class Product ซึ่งจะถูกทำเป็น Class Logger ทันใดนั้นเอง Class Product จะไม่เป็น Modular ได้อย่างสมบูรณ์ เพราะมันได้มีการเรียกใช้ Method จาก Class Logger ซึ่งไม่มีส่วนเกี่ยวข้องใดเลยกับ Class Product

AOP ก็สามารถช่วยแก้ปัญหานี้ได้ซึ่งมีการจักการอยู่ 2 – 3 อย่าง

1. ภาษที่ใช้ในการเขียน Code ปกติ จะเรียกว่า Component language
2. ภาษาที่ใช้ในการเขียน Code ส่วนที่เป็น Concern เพื่อให้ผสานกับ Component เรียกว่า Aspect language

ผลจากการนำสองภาษา Component และ Aspect มารวมกันเป็นโปรแกรม ซึ่งได้มีจัดการการทำงานบางจุดขณะโปรแกรมกำลังทำงานอยู่ บางจุดที่ว่านี้เป็นการรวมเอา

Aspect และ Component เข้าด้วยกันเรียกว่า Weave ซึ่งมันจะเกิดขึ้นเมื่อ Compile, link, run หรือ load time



การ Weave ของ AOP

AOP in AspectJ

AspectJ คือเครื่องเครื่องมือที่ช่วยในการ Implement code AOP จะอยู่ในลักษณะ Aspect language และมีคำศัพท์เฉพาะภาษาดังต่อไปนี้

- **Dynamic Join Point** (จุดเชื่อม) คือ จุดใดๆ ที่อยู่ใน Source code ที่ถูกระบุในรูปแบบที่ดี เช่น Class, Method, Exception เป็นต้น
- **Pointcut** (จุดตัด) คือ การระบุกลุ่มของ Join Point ที่มีลักษณะหรือ Concern ที่เหมือนกัน
- **Advice** (การลำดับเหตุการณ์) คือ การอธิบายพฤติกรรมของ Pointcut ใดๆ ว่าจะมีการทำงานก่อนและหลัง Pointcut อย่างไร ซึ่งมีหลายรูปแบบ เช่น Before, After, Around เป็นต้น
- **Inter-Type Declaration** (การประกาศภายในก้อนลักษณะ) คือ การสร้าง Method, ตัวแปร หรือการ Implement Interfaces ระหว่างการรันโปรแกรมอยู่ เอาไว้ใน Aspect เพื่อสามารถแก้ไขโครงสร้างของ Class นั้นๆ และความสัมพันธ์ระหว่าง Class
- **Aspect** (ก้อนลักษณะ) คือ หน่วยที่รวบรวมของการ Crosscutting Concern จะมีลักษณะคล้ายกับ Java Class ซึ่งจะเก็บเอา Pointcut และ Advice ด้วย

ตัวอย่าง ใน AspectJ ซึ่งจะเป็นการเอา Concern 2 ของตัวอย่างที่แล้วมาแสดงให้เห็น ส่วนต่างให้ได้เห็นและเข้าใจกัน

```
1: public aspect Logging {
2:        private boolean Product.confirm = false;
3:
4:        private boolean Product.assertPrice(float price){
6:                if (!confirm)
7:                        confirm = true;
8:                return (price <= 100 && price >= 0);  }

9:pointcut changePrice(Product product): target(product) && call(public * *(..));

10:before (Product product, float price): changePrice(product) && args(price){
11:               Logger logger = new Logger();
12:               logger.writeLogStart();
13:               if(product.assertPrice(price))
14:                       System.out.println("Right price: "+product.confirm);
15:               else
16:                       System.out.println("Wrong price: "+product.confirm);}
17:
18:after (Product product): changePrice(product){
19:               Logger logger = new Logger();
20:               logger.writeLogStart();
21:               System.out.println(thisJoinPoint);} }
```

บรรทัดที่1: เป็นการประกาศ Aspect ชื่อ Logging

บรรทัดที่2: เป็นการประกาศ Inter-Type declare ตัวแปร Product.confirm เป็นชนิด Boolean เพื่อให้เป็นตัวแปรที่จะบงบอกว่าได้มีการ Logging แล้ว และตัวแปร Confirm นี้ ไม่ได้ประกาศใน Class Product จริงมันจะแสดงอยู่ในเฉพาะ Aspect Logging เท่านั้น

บรรทัดที่4-8: เป็นการประกาศ Inter-Type declare ชนิด Method ชื่อ Product.assertPrice รับค่า Argument float เพื่อตรวจสอบราคาที่เปลี่ยนแปลงว่าอยู่ในช่วง 1 – 100 หรือไหม

บรรทัดที่9: Pointcut ที่รับ Argument Product เพื่อบ่งบอกเป้าหมายของ Class ที่จะรวบรวม Join point ที่ชี้ไปยังทุก Method ใน Class Product และ Class ลูก ที่ถ่ายทอดลงไป

บรรทัดที่10-16: Advice ชนิด Before ที่รับ Argument Product และ float ซึ่งเป็นตัวบ่งบอกพฤติกรรมการทำงานก่อนถึง Join Point ที่ได้ประกาศไว้ใน Pointcut changePrice

บรรทัดที่18-21: Advice ชนิด After ซึ่งเป็นตัวบ่งบอกพฤติกรรมการทำงานก่อนถึง Join Point ที่ได้ประกาศไว้ใน Pointcut changePrice

ตัวอย่าง Main Program

```java
public class Main {

    public static void main(String[] args) {
        DVD dvd =new DVD("Movie");

        dvd.putPrice(1001);
        float p = dvd.getPrice();
        System.out.println("Price = "+p);
        dvd.setStats("MV Shop",20);

    }
}
```

ตัวอย่าง ผลของการ run Program

```
1: Change Price at : 4 เม.ย. 2548, 16:04:00
2: Wrong price: true
3: Change from 0.0 to 1001.0
4: Change Price at : 4 เม.ย. 2548, 16:04:00
5: call(void com.sut.comeng.goods.DVD.putPrice(float))
6: Change Price at : 4 เม.ย. 2548, 16:04:00
7: call(float com.sut.comeng.goods.DVD.getPrice())
8: Price = 1001.0
9: Change Price at : 4 เม.ย. 2548, 16:04:00
10: call(void com.sut.comeng.goods.DVD.setStats(String, int))
```

AOP in JBossAOP

JBossAOP คือเครื่องเครื่องมือที่ช่วยในการ Implement code AOP เช่นเดียวกับ AspectJ จะอยู่ในลักษณะ XML Language ซึ่งมีคำศัพท์เฉพาะภาษาดังต่อไปนี้

- **Join Point** (จุดเชื่อม) คือ จุดใดๆ ที่อยู่ใน Source code ที่ถูกระบุในรูปแบบบที่ดี เช่น Class, Method, Exception เป็นต้น
- **Invocation** คือ Class ใน JBossAOP ซึ่งเป็นการ Encapsulate Joinpoint ต่างๆ ในขณะ runtime
- **Pointcut** (จุดตัด) คือ การระบุกลุ่มของ Join Point ที่มีลักษณะหรือ Concern ที่เหมือนกัน
- **Advice** (การลำดับเหตุการณ์) คือ การอธิบายพฤติกรรมของ Pointcut ใดๆ ว่าจะมีการทำงานก่อนและหลัง Pointcut อย่างไร ซึ่งมีหลายรูปแบบ เช่น Before, After, Around เป็นต้น
- **Aspect** (ก้อนลักษณะ) คือ หน่วยที่รวบรวมของการ Crosscutting Concern จะมีลักษณะคล้ายกับ Java Class ซึ่งจะเก็บเอา Pointcut และ Advice ด้วย
- **Introduction** คือการแก้ไขชนิดข้อมูลหรือโครงสร้างของภาษา Java มันสามารถถูกใช้เพื่อการ Implement Interface หรือการเพิ่ม Annotation

By Mr.Surachai Samattasribut
Computer Engineering School
Suranaree Univesity of Technology

- **Interceptor** คือ Aspect ที่มีแค่ Advice เดียวเท่านั้น ชื่อ invoke มันเป็นการระบุ Interface ซึ่งสามารถที่จะ Implement ถ้าต้องการให้ Code ถูกตรวจสอบโดยการให้ Class implement จาก Interface ได้เลย และมันยังนำมาใช้ใหม่ใน environment ใน JBoss อื่นๆ เช่น EJBs และ JMX MBeans

ตัวอย่าง Interceptor

```
public class Logging implements Interceptor {

        public String getName() {
                return "Test";
        }

        public Object invoke(Invocation invocation) throws Throwable {
                Logger logger = new Logger();
                logger.writeLogStart();
                try{

                return invocation.invokeNext();

                }finally{
                        logger.writeLogFinish();
                }
        }
}
```

ตัวอย่าง Aspect ที่เขียนด้วย XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<aop>
    <bind pointcut=
      "execution(public * com.sut.comeng.product.*->*(..))">
        <interceptor class="com.sut.comeng.aop.Logging"/>
    </bind>
    <bind pointcut=
      "execution(public * com.sut.comeng.goods.DVD->*(..))">
        <interceptor class="com.sut.comeng.aop.Logging"/>
    </bind>
</aop>
```

ตัวอย่าง ผลลัพธ์ที่ run โปรแกรม

```
Change Price at : 5 เม.ย. 2548, 2:57:15
Change from 0.0 to 101.0
Finish Change Price at : 5 เม.ย. 2548, 2:57:15
Change Price at : 5 เม.ย. 2548, 2:57:15
Finish Change Price at : 5 เม.ย. 2548, 2:57:15
Price = 101.0
Change Price at : 5 เม.ย. 2548, 2:57:15
Finish Change Price at : 5 เม.ย. 2548, 2:57:15
```

AspectJ VS JBossAOP

AspectJ และ JBoassAOP ต่างก็เป็นเครื่องมือเอาไว้สำหรับ Implement AOP ได้เหมือนๆ กัน ต่างกันตรงที่ลักษณะของโครงสร้างของภาษา และวิธีการเรียกใช้ ต่อไปนี้จะเป็นการนำคำศัพท์ของทั้ง AspectJ และ JBossAOP ซึ่งมีความหมายเหมือนกัน

AspectJ	**JBossAOP**
Jointpoint	Joinpoint
Pointcut	Invocation, Pointcut
Advice	Advice
Aspect	Interceptor
Inter-type Declare	Introduction

By Mr.Surachai Samattasribut
Computer Engineering School
Suranaree Univesity of Technology

Conclusion

AOP เป็นแนวความคิดที่จะเอามาช่วยในการเขียน Code แบบ OOP เท่านั้น มิใช่จะมาแทนที่ OOP โดยสิ้นเชิง ซึ่งมีเครื่องมือหลายอย่างที่นำมาช่วย Implement AOP นอกจากที่กล่าวมาแล้วก็ยังมีอีกมากมาย อาทิเช่น AspectWerkz เป็นต้น จากเรียนรู้และทดลองใช้ AOP ก็ยังมีข้อที่ไม่เข้าใจหลายประการ ซึ่งต้องใช้เวลาเป็นอย่างมากถึงจะเข้าใจอย่างถ่องแท้ ไม่ว่าจะเป็นคำสั่งที่อยู่ใน AOP หรือในเครื่องมือต่างๆ แต่มันก็ยังจำเป็นต้องศึกษาต่อไป เพราะมันมีประโยชน์อย่างมาก ทั้งด้านการจัดเรียง Code ให้เป็นระเบียง ละก็ยังสามารถนำมาใช้อีกด้วย

By Mr.Surachai Samattasribut
Computer Engineering School
Suranaree Univesity of Technology

Reference

http://www.eclipse.org/ajdt
http://www.eclipse.org/aspectj
http://www.jboss.org/products/aop
http://aspectj.org
http://docs.codehaus.org/display/AW/Eclipse+plugin

By Mr.Surachai Samattasribut
Computer Engineering School
Suranaree Univesity of Technology