COMPSCI 732
Software Tools and Techniques

Lecturer:
Dr SANTOKH SINGH

Lecture 1: Advanced topics in Aspect Oriented Development

Contact Details

- Email: santokh@cs.auckland.ac.nz
- Office: RM 488, Level 4, new CS building
- Phone: ext 82283
- Office Hours: Monday(11am - 12pm) *or see me anytime mutually convenient.*

Overall Lectures Description

Lecture 1: Aspects and AOP

Lecture 2: Aspect Oriented Systems and Components

Lecture 3: Support Tools

Lecture 4: Aspect Oriented Software Development across the Software Development Lifecycle & Review

Aspects

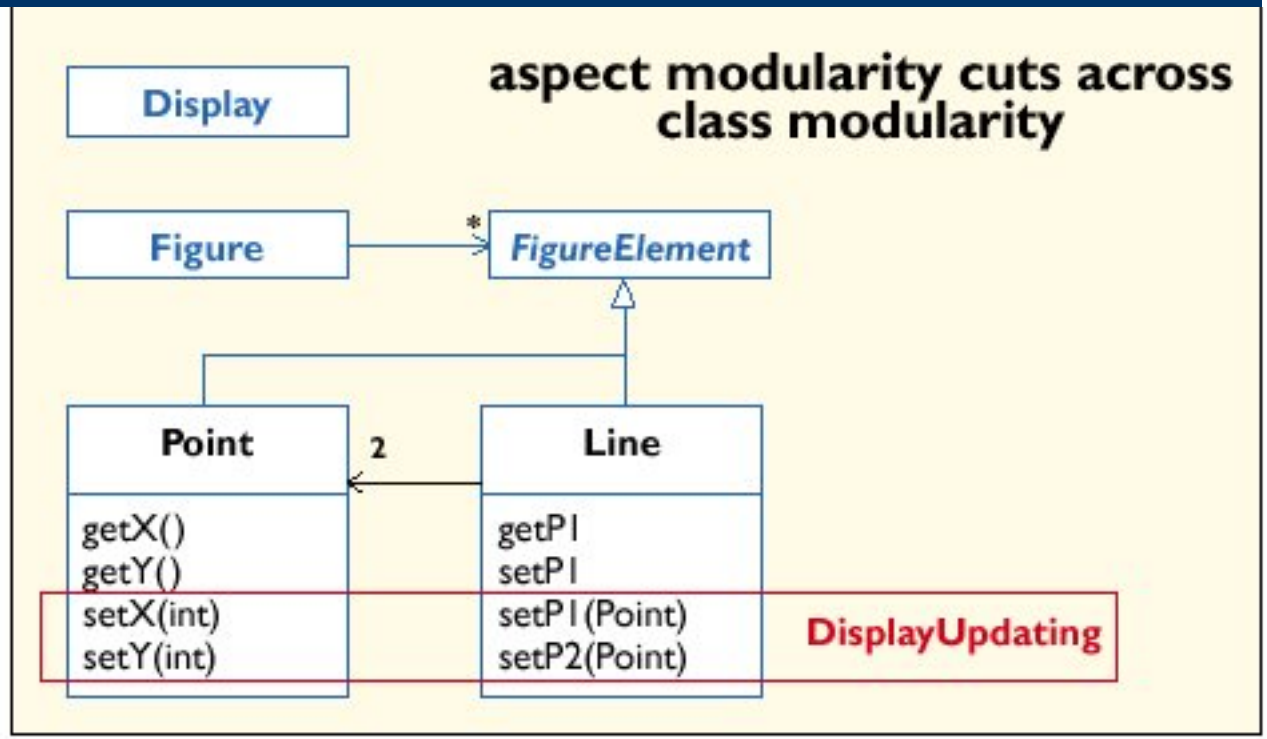
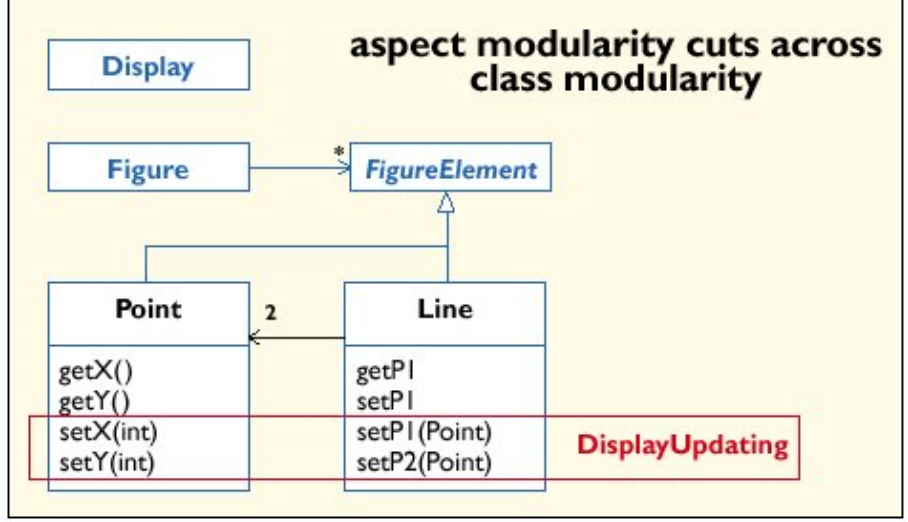


Figure 1: **Aspects crosscutting classes in a figure editor**

Aspects

- Gregor Kiczales and his research group at Xerox PARC conceived the idea of using aspects in Aspect-Oriented Programming (AOP) to address cross-cutting issues that were spread out in the designs and implementation code of software systems.
- These cross-cutting issues “mangled” the system’s functionalities making them hard to understand, modify and control.
- In some cases, ultimately, remaining as redundant code because no one wants to delete, edit or to have anything to do with it because understanding and dealing with it might take up too much time and resources.
- Neither procedural nor object-oriented programming techniques are sufficient to clearly capture these cross-cutting issues in software designs and implementations and neither of these programming techniques can be used to address cross-cutting issues.
- Concerns are said to crosscut if the methods related to those concerns intersect. AOP deals with crosscutting concerns and their descriptions, designs and implementations.

Aspects



- As an illustration for understanding more about aspects, we consider a UML diagram for a simple figure editor, as depicted in figure 1, in which there are two concrete classes called the Point and Line classes of the FigureElement super class

- These classes manifest good modularity, in that the source code in each class is closely related and each class has a clear and well-defined interface. But consider the concern that the screen manager should be notified whenever a figure element moves.

- Every method that moves a figure element requires notification to get the display updated (DisplayUpdating).

Aspects

- The red box in figure 1 is drawn around every method that must implement this DisplayUpdating concern, just as the Point and Line boxes are drawn around every method that implements this cross-cutting concern.

- Notice that the box for DisplayUpdating behaviour fits neither inside of nor around the other boxes in the figure, instead it cuts across the other boxes. This is why we call it a crosscutting concern. The bigger the application, the more pronounced and cluttered the cross-cutting issues become.

- Using just Object Oriented programming, the implementation of crosscutting concerns tends to be scattered throughout across the system, just as it is shown in the figure above. But by using the mechanisms of AOP, the implementation of the DisplayUpdating behaviour can be captured and modularised into a single aspect.

- Since we can implement this behaviour in a single modular unit, it makes it easier for us to think about it as a single design unit. In this way, having the programming language mechanisms of aspects lets us think in terms of aspects at the design level as well.

Aspects and AOP

- These cross cutting issues are called aspects, and other terms like tangling, intermingling, mangling and interleaving units have also been used to describe them.

- A better description of aspects is attributed to Gregor Kiczales where he states that an aspect is a modular unit that cross cuts the structure of other modular units and that it can encapsulate state, behaviour and behaviour enhancements in other units.

- The goal of AOP is to make designs and code more modular, meaning the concerns are more localized using AOP rather than scattered, and have well-defined interfaces with the rest of the system.

- AOP provides us with the benefits of modularity, including making it possible to reason about different concerns in relative isolation, making them pluggable and amenable to separate parallel development during the software development process.

Aspects and AOP

- There are two types of aspects, design aspects and program (or code) aspects. Modular units of design that cross-cut the structure of other parts of the design are called design aspects. Similarly modular units of programs that cross-cut other modular units of programs are called aspects. Both these types of aspects give rise to cross-cutting issues in designs and implementations.

- The AOP methodology formulated techniques to solve cross-cutting issues and these include isolation of aspects, reuse of aspect code and composition of aspects from the onset. AOP also uses an approach which is called code weaving to tackle cross-cutting issues in programs. This technique is carried out by composing the aspects properly, identifying the regions where the aspects appear and weaving the aspects into the regions so as to produce the desired results.

- AOP also clearly distinguishes components from aspects. In this methodology, a component can be cleanly encapsulated in a generalised procedure. Furthermore components tend to be units of the systems functional decompositions. For instance, in a collaborative travel planner system, the booking and system customer are both components because they can be both cleanly encapsulated in a generalised procedure and are units of the systems functional decomposition.

Aspects and AOP

- A very popular AOP technology is AspectJ. *(We WILL use this in our assignment).*
- AspectJ is actually an aspect-oriented extension to the Java programming language and the Xerox PARC group's work is now integrated into the currently popular Eclipse Java IDE.
- There are 3 basic constructs in AspectJ, i.e. the join points, pointcuts and advice. The joint points are the points where the crosscutting occurs, a pointcut defines a set of execution points for the joint points and the advice represents what to do in the cross-cutting area.
- Other commercial Aspect-oriented frameworks include JBoss, AspectWerkz and Spring AOP. All these have also helped popularise AspectJ that has become one of the most widely-used aspect-oriented languages to address cross-cutting issues. AOP techniques have been used in other languages and platforms as well.
- AOP has been applied to metadata and their interceptors in the Aspect Builder application, where services between clients and other objects were stacked semi-seamlessly in COM and seamlessly in .NET.
- Microsoft has also announced that it has been developing a state-of-the-art aspect-oriented programming tool called Aspect.NET which can be integrated with the latest Visual Studio IDE. In this project Microsoft aims to make AOP ubiquitous for .NET software engineers, develop the most adequate ways of representing aspects and lay the foundation for future research and development work on spreading AOP among .NET users.
- The Aspect-Oriented Programming methodology together with ideas from its vast research carried out so far has contributed immensely to our understanding of these cross-cutting modular units called aspects in design and implementation. They have also helped software developers gain a better understanding in separating components from aspects.

Homework – *please do!*

- Read up more about the concepts associated with the following terms in AspectJ:
  - **Join point**: well defined points in the execution of a program.
  - **Pointcut**: collection of join points.
  - **Advice**: a construct indicating code that should run at each join point in a pointcut. *What are the different types of Advice?*
  - **Aspect**: program units encapsulating an implementation of a cross cutting property.

A Few Points to Remember

- The AOP mechanism let us write an aspect as a separate, localized, modular piece of code.
- So we can think about them as separate units. For hard-core software designers/coders: we can think of each aspect as being in its own box.
- AOP is NOT a replacement for Object-Oriented Programming (OOP) – AOP builds on OOP to provide techniques for solving problems for which OOP is insufficient.