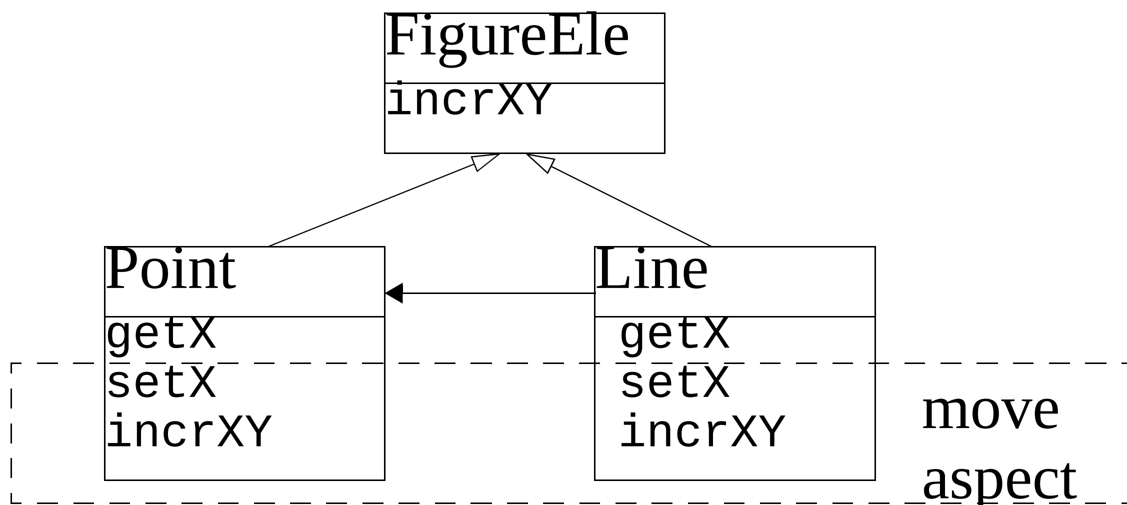
AspectJ

- A very nice MOP/general compositional semantic extensibility facility for Java
  - used entirely for interposing code, not changing how the object system works
  - AspectJ is a transparent extension of Java, comes with IDE support (for easier editing, inspection of aspect code)
- To demonstrate, consider an example application: a figure editor



Join points

- Many possible join points in AspectJ. At:
  - method call (inside calling object)
  - method call reception by an object (any method)
  - method execution (specific method)
  - field access (get/set)
  - constructor call (inside object doing `new`)
  - constructor call reception (any constructor)
  - exception handler execution
  - class initialization (static initializers run)

Pointcuts

- *Pointcut* = set of join points + values from the context (e.g., the `this` object, method parameters, etc.)

```
call(void Point.setX(int))
```

  - all join points where the method called is `void Point.setX(int)`

Kinds of Pointcuts

- Pointcuts can be thought of as runtime predicates: when they are true, we are at a join point described by the pointcut.
- Several kinds of pointcuts. E.g.:
  - `call(`*`signature`*`)`
  - `execution(`*`signature`*`)`
  - `get/set(`*`signature`*`)`
    - value can be matched with `args`
  - `args(`*`Type`*`)`
  - `handler(`*`ThrowableClass`*`)`
  - `this/target(`*`Type`*`)`
  - `within(`*`Type`*`)`
  - `withincode(`*`signature`*`)`
  - `cflow(`*`pointcut`*`)`
  - `initialization(`*`ConstrSig`*`)`
  - `staticinitialization(`*`Type`*`)`
- Also: boolean pointcut operators (&&, `||`, etc.) and pointcut constants (user-defined pointcuts)

Pointcut Example

```
pointcut moves():
   call(void FigureElement.incrXY(int,int))
|| call(void Line.setP1(Point))
|| call(void Line.setP2(Point))
|| call(void Point.setX(int))
|| call(void Point.setY(int));
```

- describes the join points where methods that cause “movement” of a figure are called
  - Note that a “user-defined” pointcut (operator `pointcut`) is used to give a name (`moves`) to the pointcut

Advice

- *Advice*: specification of aspect code to be interposed at pointcuts
  - before, after, or instead of (`around`) the code at a join point
    - two special cases of “after”: after returning/after throwing (for normal/exception exits)

Aspects

- Aspects have class-like syntax (and, to some extent, semantics—e.g., for scoping). They can contain pointcuts, advice, and regular class declarations (member vars/methods)

```
aspect MoveTracking {
  static boolean flag = false;
  static boolean testAndClear() {
    boolean result = flag;
    flag = false;
    return result;
  }

  pointcut moves():
   call(void FigureElement.incrXY(int,int))
   || call(void Line.setP1(Point))
   || call(void Line.setP2(Point))
   || call(void Point.setX(int))
   || call(void Point.setY(int));

  after(): moves() {        // advice
    flag = true;
  }
}
```

Aspects

- Aspects can have multiple instances
- There are complex rules about how aspect execution (advice application) is ordered
  - the rules take into account Aspect relationships (e.g., if aspect A extends B, then it's considered more specific)
  - there is a `dominates` keyword for aspects that know about each other

Example (uses `MoveTracking` from last slide)

```
aspect Mobility dominates MoveTracking {
  static boolean enableMoves = true;

  around() returns void:
    MoveTracking.moves()
  { if (enableMoves) proceed(); }
}
```

defines an "around" (instead-of) method preventing moves if the flag is not set

Pointcut Parameters

- Advice and pointcut definitions can have parameters (see empty parentheses in previous examples)
- The parameters can be used in pointcut predicates instead of type variables and take the value *of the instance matching the predicate*
  - this is overloading the existing syntax for an entirely different purpose

```
before(Point p, int nval):
  call(void p.setX(nval)) {
    System.out.println("x value of" + p +
      " will be set to " + nval + ".");
  }
```

To print a message every time the value of `x` for a point changes

Example: Getting the current object

- regular pointcut definition:
```
pointcut foo() :
  instanceof(Point);
```
- pointcut with parameter:
```
pointcut foo(Point p) :
  instanceof(p);
```
- `p` is the object of class `Point` with which the join point is associated!

Example: Around Advice and Proceed

- We saw `proceed` earlier, but it can also be called with parameters
- To ensure that a method is only called with non-negative int arguments:
```
around(int nv) returns void:
 call(void Point.setX(nv))
{ proceed(Math.max(0, nv)); }
```

Abstract and Generic Aspects

A “virtual type”-like mechanism allows aspect genericity

```
abstract aspect SimpleTracing {
  abstract pointcut tracePoints();
  //yet undefined

  before(): tracePoints() {
    printMessage(“Entering”,thisJoinPoint);
  }
  after(): tracePoints() {
    printMessage(“Exiting”,thisJointPoint);
  }

  void printMessage(String s, JoinPoint tjp)
  { ... }
}

aspect XYTracing extends SimpleTracing {
  pointcut tracePoints():
    call(
      void FigureElement.incrXY(int,int));
}
```

- (note the `thisJointPoint` variable and the `JoinPoint` type: they reflectively export details of the AspectJ implementation)

Wildcards

E.g.,

```
call(* Point.*(..))
call(Point.new(..))
```

Control-Flow Based Pointcuts

The `cflow` operator is true on points under the dynamic extent of other join points (e.g., while the methods corresponding to these join points are still active on the execution stack)

```
pointcut moves(FigureElement fe):
<see before>;

pointcut topLevelMoves(FigureElement fe):
 moves(fe) && !cflow(moves(FigureElement));
```

Implementation

The AspectJ compiler inserts code to check and call the right aspects at join points: efficient

Introductions / Inter-type Declarations

Can declare members and supertypes for existing classes!

A static transformation language. These "introductions" are not advice and are not associated with pointcuts

Add an "`enabled`" field to all `FigureElement`s:

```
- boolean FigureElement.enabled=false;
```

Add a setter method:

```
- public
   FigureElement.setEnabled(boolean b) {
    this.enabled = b;
   }
```

Add superclasses to FigureElement:

```
- declare parents:
    FigureElement extends Drawable
```

Overall Critique

- AspectJ is a good tool, but not particularly ground-breaking
- The question is, how much “aspect”-functionality is MOP-like?
    - probably not much:
        - most of the compositional functionality (e.g., before-after methods) can be done without MOPs (e.g., with mixins)
        - the rest needs a full blown generator