Using Aspect-Oriented Programming to extend Protégé

Henrik Eriksson

Linköping University

Questions about MOP and Protégé

- Original goal: Extending the JessTab plug-in
- What is the class precedence in Protégé? Really?
- Where is the source code for computing the class precedence list?
- Difficult question for several reasons:
  - Protégé source code not documented
  - Code is blaming other parts of the code (sometimes called OO-design)
  - Protégé source code not commented
  - Protégé source code not commented

LINKÖPINGS UNIVERSITET

ClosureUtils.calculateClosure()

- Finally, the most fundamental method—the essence of Protégé…

```
 public static Set calculateClosure(
     BasicFrameStore store,
     Frame frame,
     Slot slot,
     Facet facet,
     boolean isTemplate) {
     return calculateClosure(store, frame, slot, facet, isTemplate, new LinkedHashSet());
 }

// TODO It would be preferable if this method returned a breadth first closure
 private static Set calculateClosure(
     BasicFrameStore store,
     Frame frame,
     Slot slot,
     Facet facet,
     boolean isTemplate,
     Set values) {
     Iterator i = store.getValues(frame, slot, facet, isTemplate).iterator();
     while (i.hasNext()) {
         Object o = i.next();
         boolean changed = values.add(o);
         if (changed && o instanceof Frame) {
             calculateClosure(store, (Frame) o, slot, facet, isTemplate, values);
         }
     }
     return values;
 }
```

Examining the code: // TODO…???

- Wait, there is a comment here. Ray is speaking to us!

// TODO It would be preferable if this method returned a breadth first closure

Extending Protégé

- Protégé extensions
  - Major strength of the Protégé architecture
  - Community-based development
- Several different ways of extending Protégé
  - Tab, widget, and backend plug-ins
  - Replacing the knowledge-base model
  - Modifying Protégé source code
- Modifying ClosureUtils.calculateClosure()
  - Cannot be accomplished through the API
  - Requires source-code changes
  - Results in version-control issues

Aspect-Oriented Programming (AOP)

- Problem: Some issues are not well captured by traditional programming methodologies
  - Often, issues *cut across* the natural units of modularity
  - Examples: Error handling, logging, security
- Solution: Modularize crosscutting concerns through aspect-oriented programming
  - Just like object-oriented programming modularizes common concerns
  - Extension of object-oriented programming
- Aspect-oriented programming for Java: AspectJ
  - http://www.eclipse.org/aspectj/

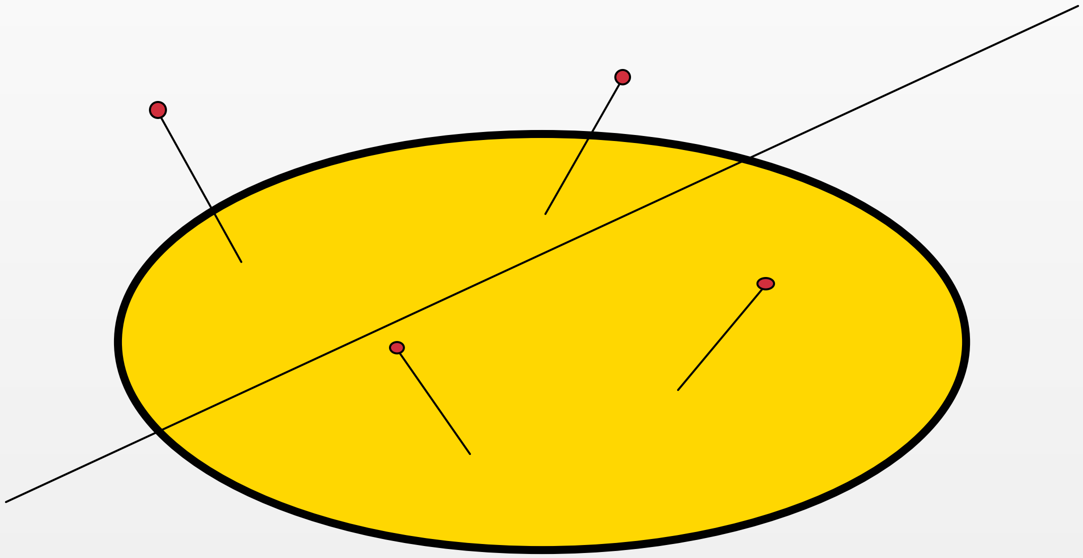
AspectJ

Key concepts:

- Join point – A well-defined point in the program flow
- Pointcut – A way of selecting certain join points
- Advice – The code to execute when a point cut is reached
- Introduction – Modification of the static structure of the program (e.g., introduction of members)
- Aspect – Unit of modularity for crosscutting concerns
- Weaving – The process of “compiling” in AOP

AspectJ

Pointcuts



Pointcuts

- Name-based crosscutting
- The pointcut

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

  identifies any call to the method setX defined on Point objects

- Pointcuts can be composed, for example:

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

Wildcard pointcuts

Property-based crosscutting

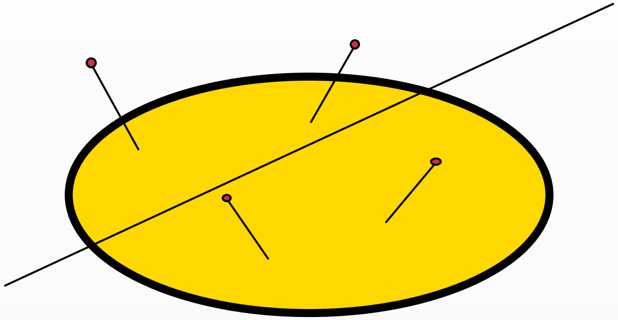
- call(void Figure.set*(..))
  - Calls to methods on Figure that begin with “set”
- call(public * Figure.* (..))
  - Calls to any public method on Figure

The operator cflow

- identifies join points that occur in the dynamic context of another pointcut

- cflow(move())
  - all join points that occur “inside” (when calling) methods in move

Advice



- What to do when you reach a pointcut
- Additional code that should run at join points
- Advice types
  - Before
  - After
  - Around

```
after(): move() {
   System.out.println("A figure element was moved.");
}
```

Called after move join points

Accessing execution context in pointcuts

- Example: Print the figure element that was moved and its new coordinates after a call to setXY

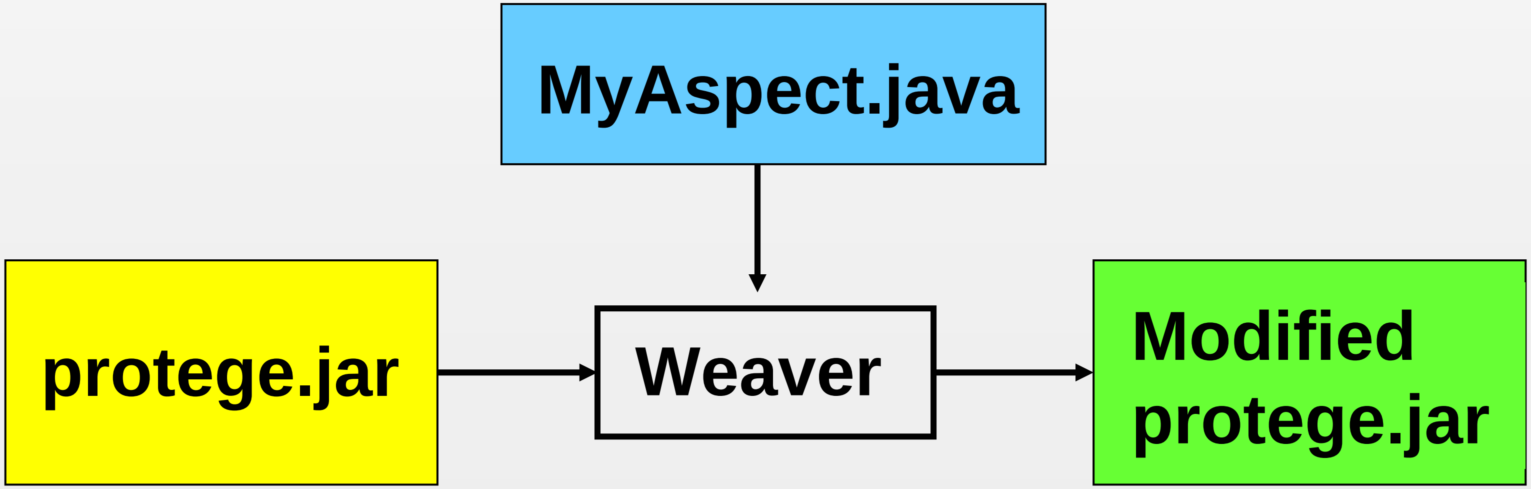
```
pointcut setXY(FigureElement fe, int x, int y):
   call(void FigureElement.setXY(int, int))
   && target(fe)
   && args(x, y);

after(FigureElement fe, int x, int y): setXY(fe, x, y) {
   System.out.println(fe + " moved to (" + x + ", " + y + ").");
}
```

Uses of AOP

- Tracing, logging, profiling
- Pre- and post-conditions
  - Checking arguments and return values
- Contract enforcement
  - Identify method calls that should not exist
- Configuration management
  - Different version of the same program by including different aspects
- Modifying existing code
  - High-level “patching” language
  - Can weave on source and compiled code (e.g., jar files)
  - Load-time weaving in the future

AOP and Protégé

- Extending/modifying Protégé
  - Protégé API and GUI
  - Preexitsing plug-ins
- Weaving aspects with protege.jar

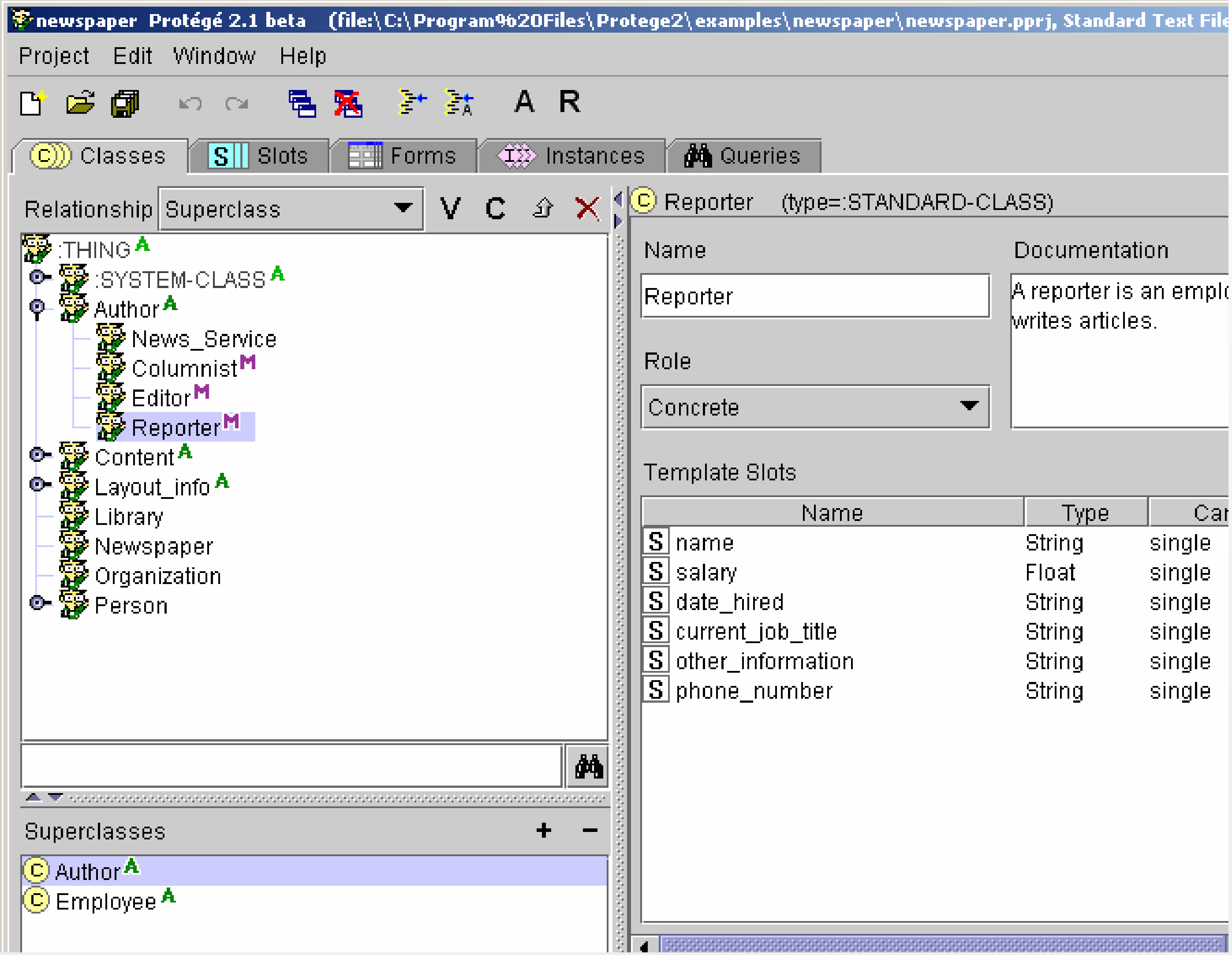


Example 1: GUI Skin

- Creating a skin for Protégé
- Replace the class icon in the class tree

```
aspect Skin {

    after() returning(FrameRenderer x) :
          execution(Component DefaultRenderer.getTreeCellRendererComponent(..)) {
                x.setMainIcon(Icons.getNerd16x16Icon());
    }

}
```

Result: Protégé with aspect Skin



Relationship: Superclass

Reporter (type=:STANDARD-CLASS)

Name: Reporter

Documentation: A reporter is an emplo writes articles.

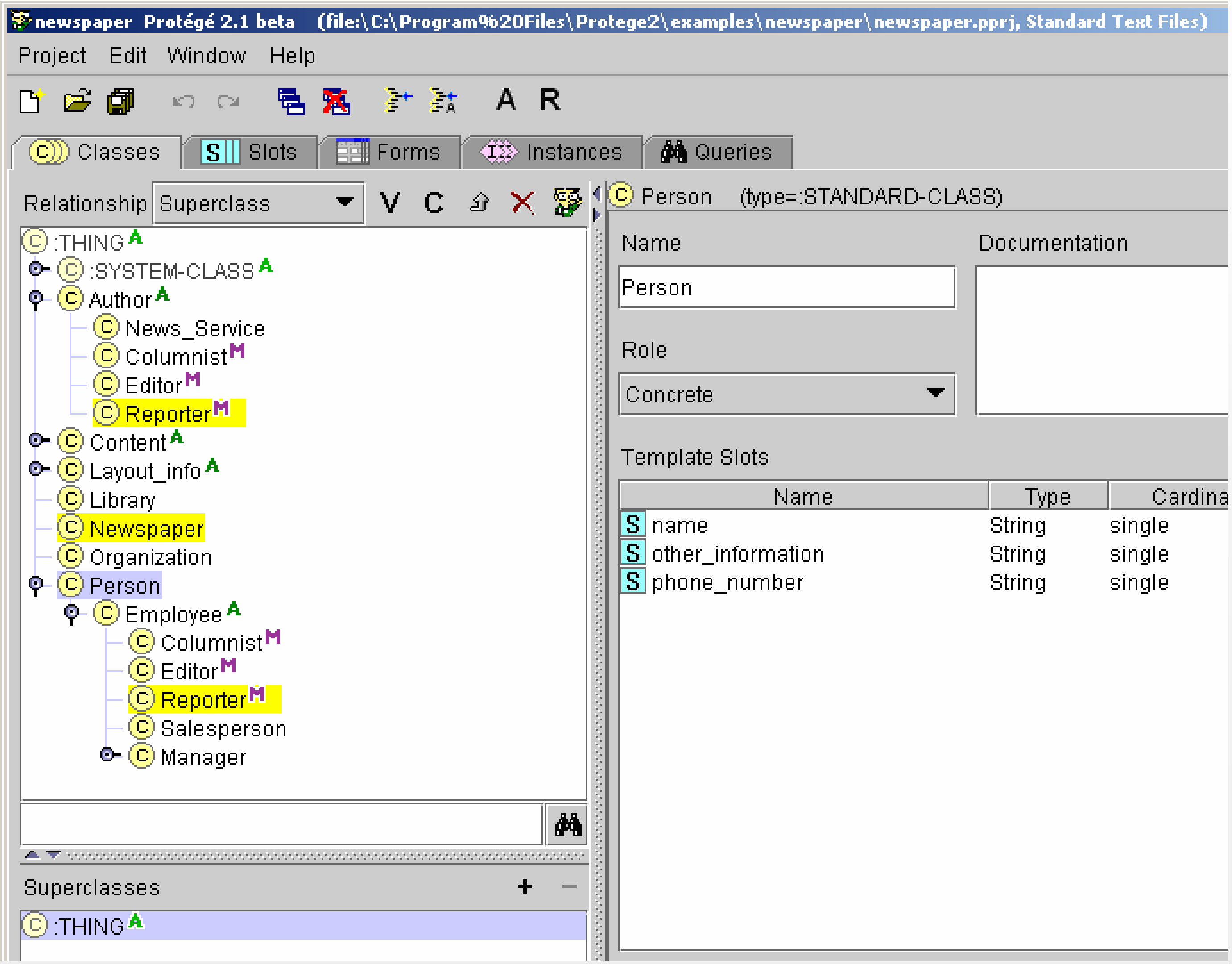
Role: Concrete

	Name	Type	Car
S	name	String	single
S	salary	Float	single
S	date_hired	String	single
S	current_job_title	String	single
S	other_information	String	single
S	phone_number	String	single

Example 2: Yellow Marker

```
privileged aspect YellowMarker {
  after(ParentChildNode value) returning(FrameRenderer x) :
      args(*, value, ..) && execution(Component DefaultRenderer.getTreeCellRendererComponent(..)) {
           if (value.getCls().isYellow()) {
             x._backgroundNormalColor = Color.yellow;
             x._backgroundSelectionColor = Color.yellow.darker();
           }
  }
  after(final ClsesPanel cp) : target(cp) && execution(ClsesPanel.new(..)) {
           cp._labeledComponent.addHeaderButton(
               new AllowableAction("Mark selected class as yellow", Icons.getNerd16x16Icon(), cp) {
                      public void actionPerformed(ActionEvent event) {
                        for (Iterator i = getSelection().iterator(); i.hasNext(); ) {
                                 Cls c = (Cls)i.next();
                                 c.setYellow(!c.isYellow());
                        }
                        cp.repaint();
                      }
             });
  }
  private boolean Cls._yellow = false;
  public boolean Cls.isYellow() { return _yellow; }
  public void Cls.setYellow(boolean flag) { _yellow = flag; }
}
```

Result: Protégé with Yellow Marker



newspaper Protégé 2.1 beta (file:\C:\Program%20Files\Protege2\examples\newspaper\newspaper.pprj, Standard Text Files)

Relationship: Superclass

```
:THING A
  :SYSTEM-CLASS A
  Author A
    News_Service
    Columnist M
    Editor M
    Reporter M
  Content A
  Layout_info A
  Library
  Newspaper
  Organization
  Person
    Employee A
      Columnist M
      Editor M
      Reporter M
      Salesperson
      Manager
```

Superclasses

:THING A

Person (type=:STANDARD-CLASS)

Name: Person

Documentation:

Role: Concrete

Template Slots

	Name	Type	Cardina
S	name	String	single
S	other_information	String	single
S	phone_number	String	single

Example 3: Controlling the class precedence list in Protégé

```
pointcut computePrecedence(Frame frame, Slot slot, Facet facet, boolean
     isTemplate, ClosureCachingBasicFrameStore target): target(target) && if
     (frame.getProject() != null && slot.getFrameID() ==
     Model.Slot.ID.DIRECT_SUPERCLASSES) && args(frame, slot, facet, isTemplate)
     && execution(Set ClosureUtils.calculateClosure(BasicFrameStore, Frame, Slot,
     Facet, boolean));


Set around(Frame frame, Slot slot, Facet facet, boolean isTemplate,
     ClosureCachingBasicFrameStore target) : computePrecedence(frame, slot, facet,
     isTemplate, target) {

  // Compute custom class precedence list here and return the result

 }
```

LINKÖPINGS UNIVERSITET

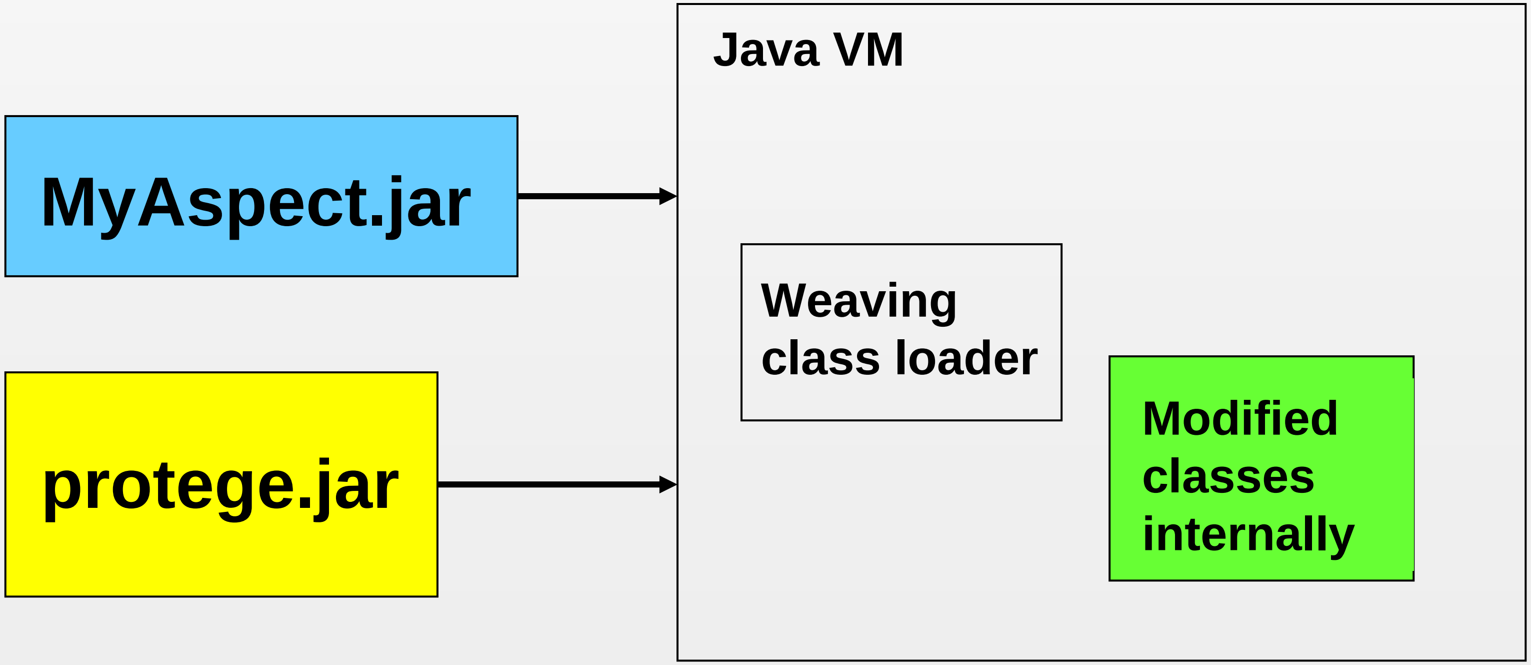
Load Time Weaving

- Class loader weaving
- Replaces standard class loader
- Slightly slower class load time
- Available in AspectJ 1.2
- Works with the core Protégé system
- Affected plug-ins must be on classpath as startup
  - Some differences in class-loading approaches
  - Set with -Daj.class.path=

LINKÖPINGS UNIVERSITET

Load Time Weaving and Protégé

- Special startup script required
- Select aspect(s) at startup



Summary

- AOP and AspectJ
  - Are cool techniques
  - Allows for powerful modifications
  - Removes the problems of modifying source code
  - Support load-time weaving
- Protégé works well with AspectJ
  - Different flavors of Protégé depending on the aspects used
  - Aspects that complement plug-ins possible