



Software Development with CaesarJ

Software Engineering
Design

Refinement & composition

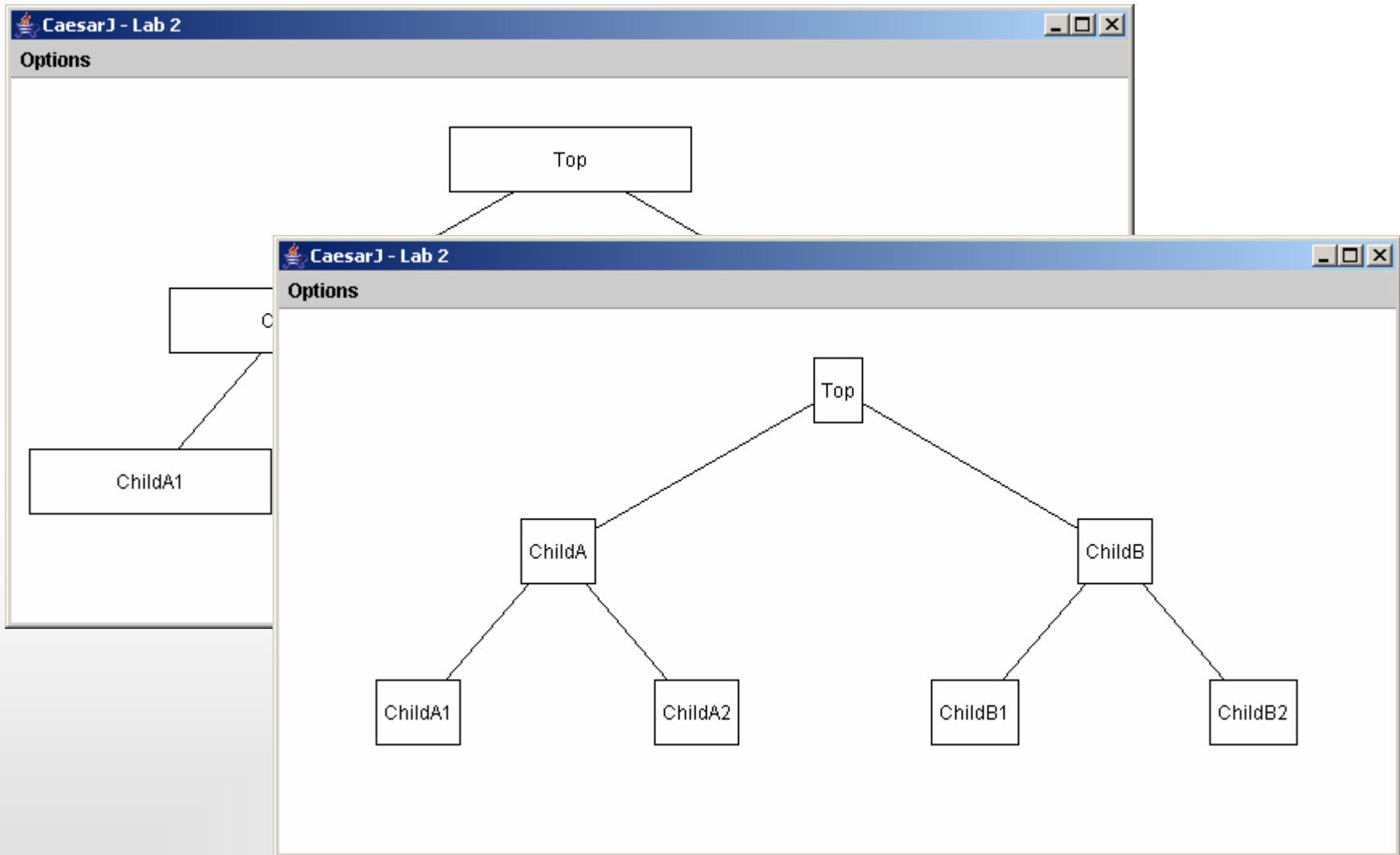
- Virtual classes
- Propagating mix-in composition
- Dependent types

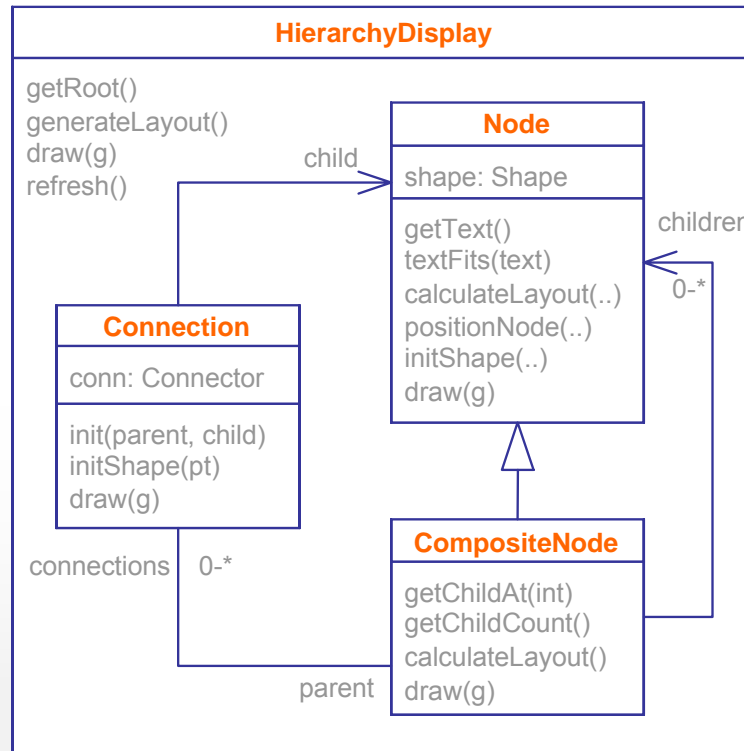
Integrating crosscutting concerns

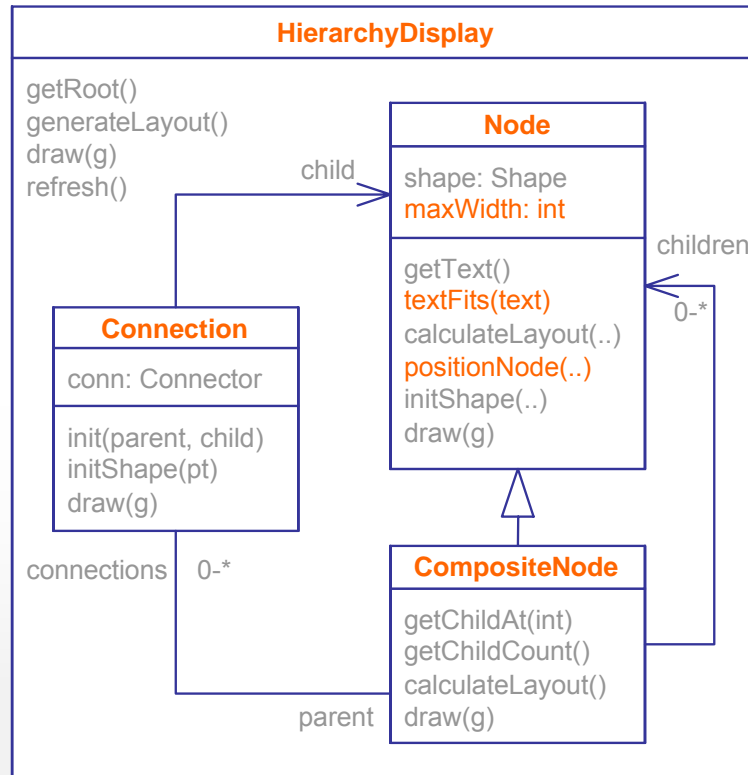
- AspectJ style pointcuts
- Dynamic aspect deployment
- Bindings

- **Component Extension**
 - Extending component with virtual classes
 - Combining different extensions
- **Component Integration**
 - Defining wrapper classes
 - Observing events with pointcuts
 - Dynamic aspect deployment
- **Variability Management**
 - Varying component implementations
 - Varying component bindings
 - Extending collaboration interface
- **Integrating Distributed Components**

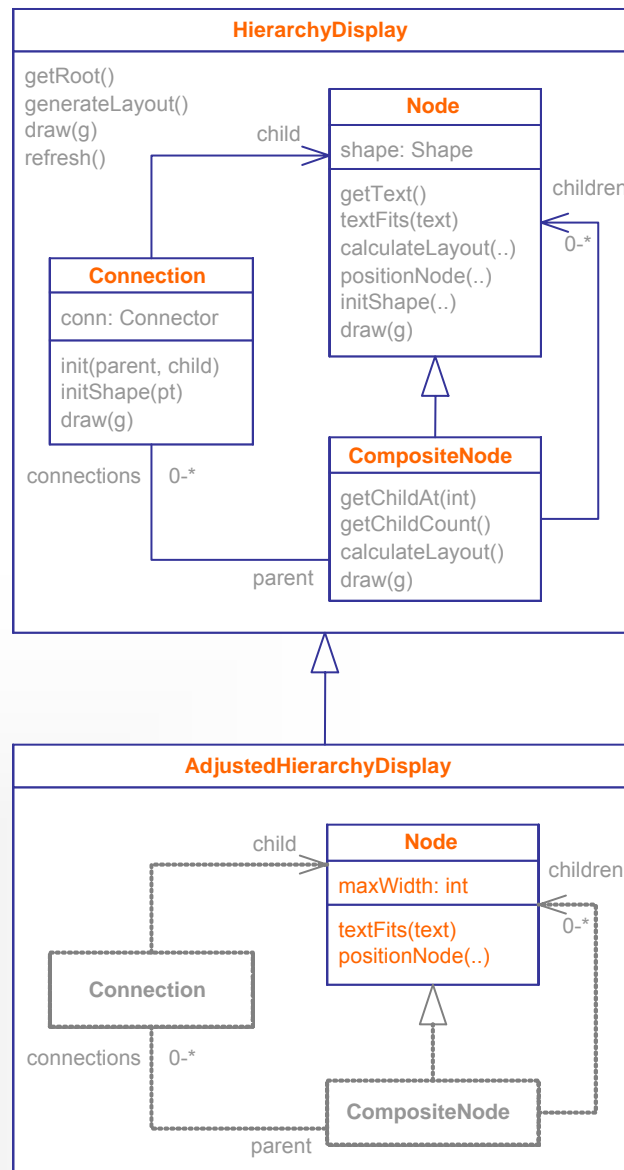
- **Component Extension**
 - Extending component with virtual classes
 - Combining different extensions
- Component Integration
 - Defining wrapper classes
 - Observing events with pointcuts
 - Dynamic aspect deployment
- Variability Management
 - Varying component implementations
 - Varying component bindings
 - Extending collaboration interface
- Integrating Distributed Components



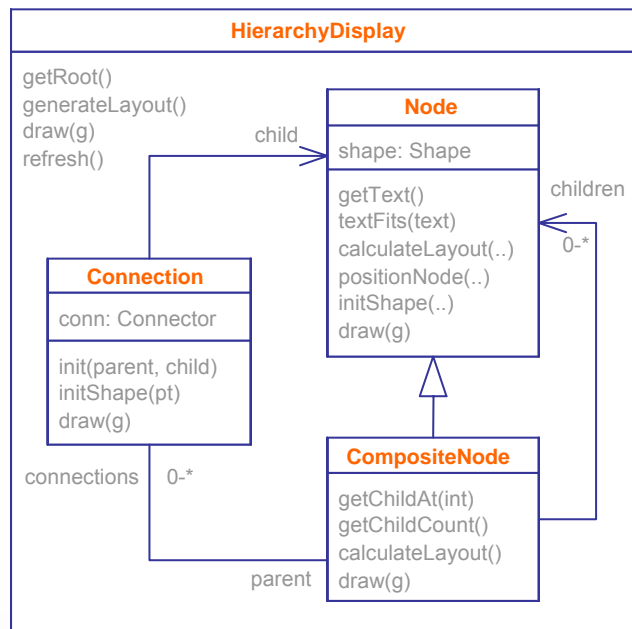




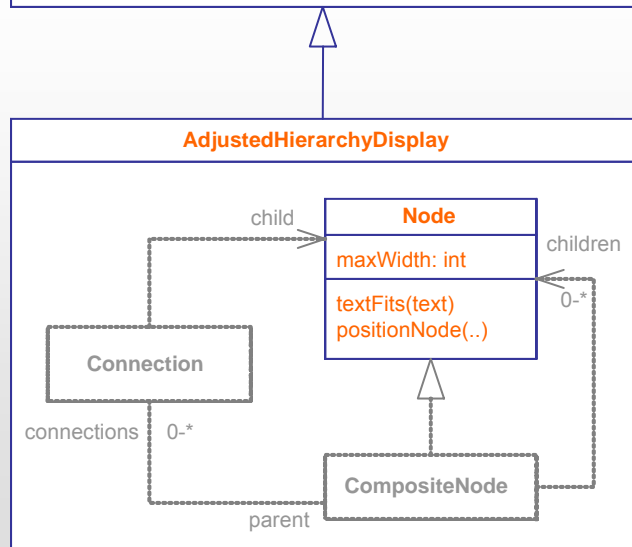
<caesarj> Virtual Classes

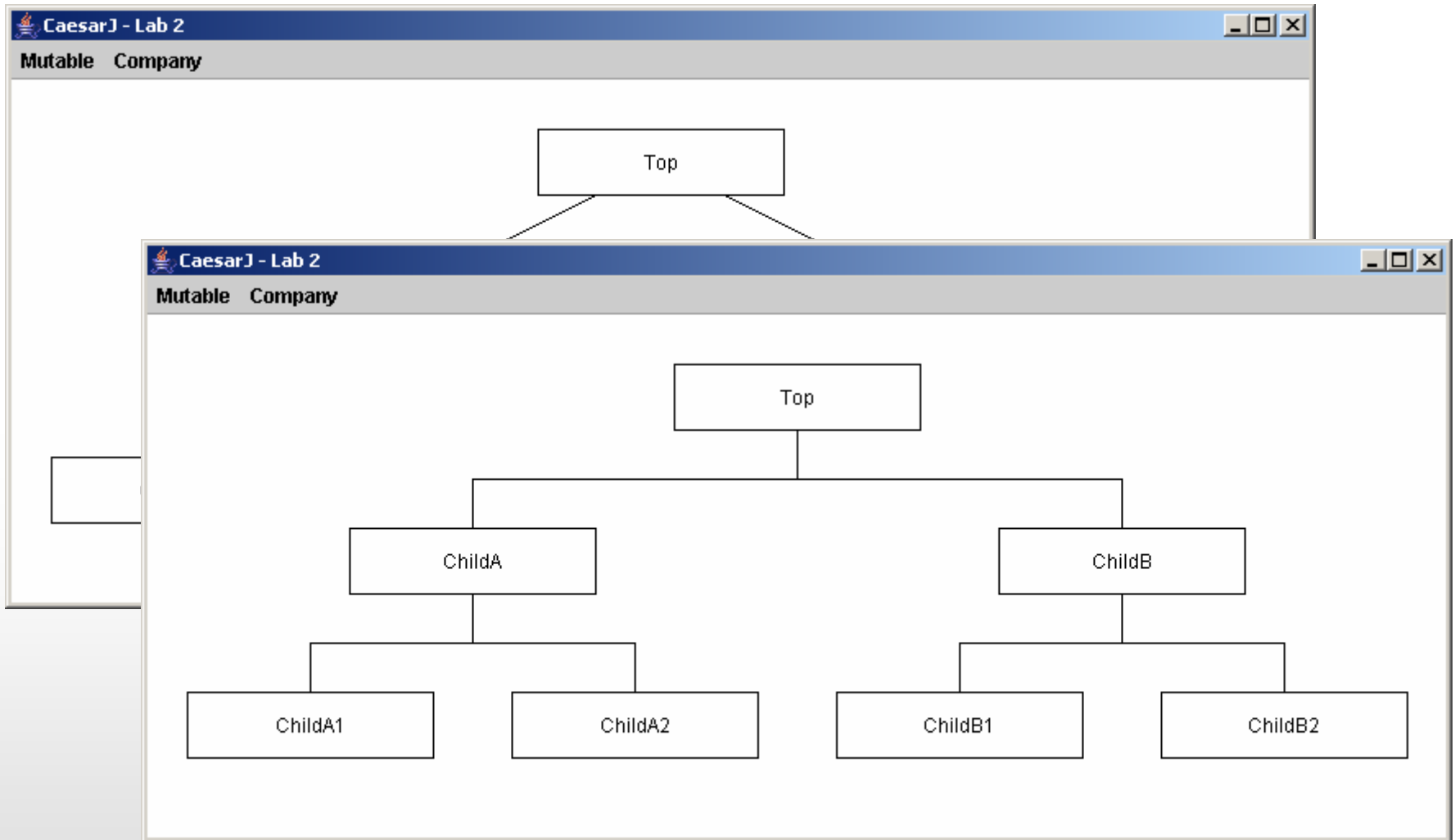


<caesarj> Virtual Classes



- Can be overridden and late bound (just like virtual methods)
- Old relations are inherited but automatically re-directed to the most specific definition of a type reference
- New classes and relations can be added





<caesarj> To Do: Change Connector Type

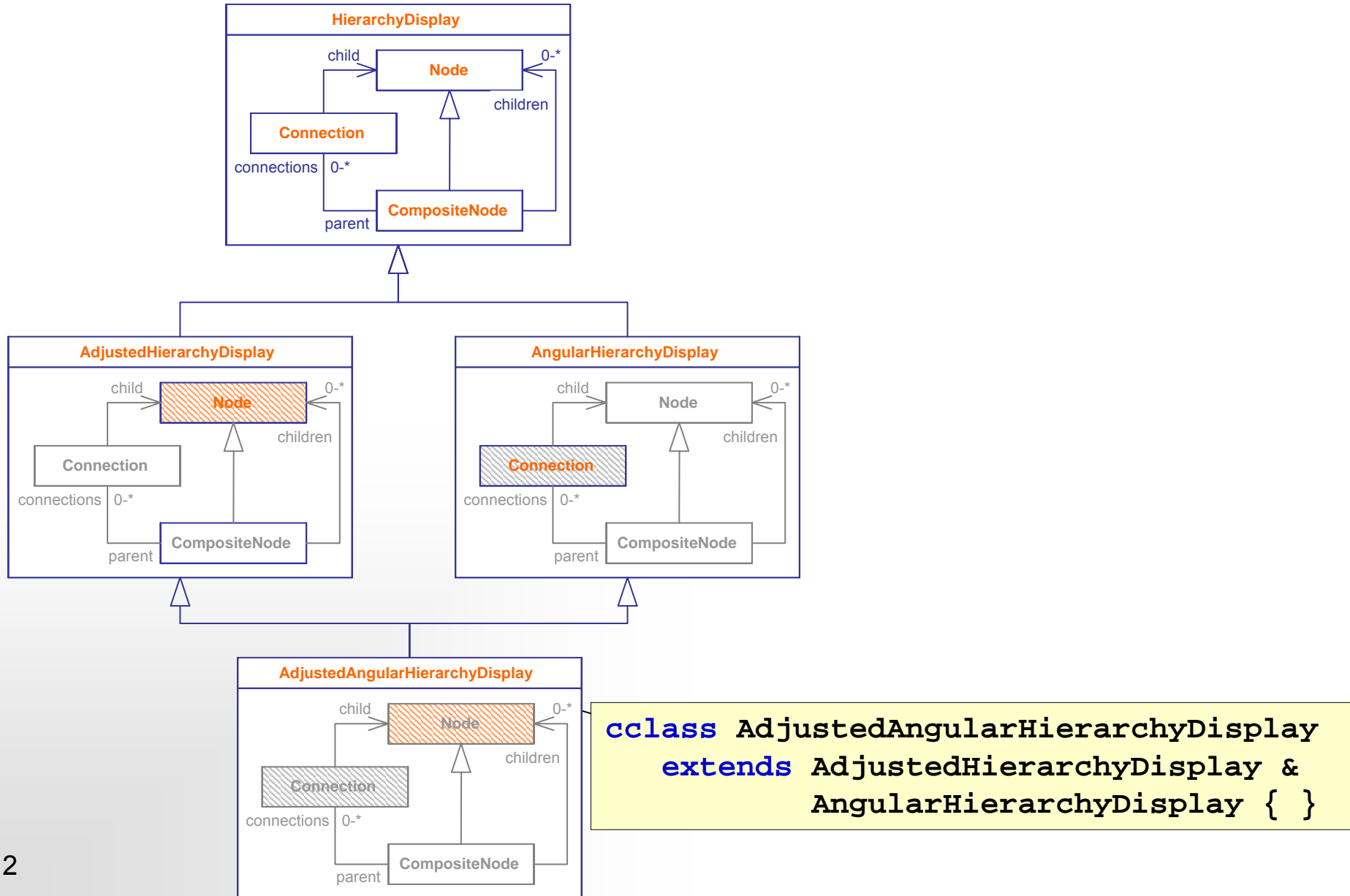
Task:

Extend hierarchy display component to use angled connectors.

Steps:

1. Open `Lab2A`
2. Declare `AngularHierarchyDisplay` as extension of `HierarchyDisplay`
3. Override `Connection` class
4. Override `initShape()` method
5. Implement it using `RightAngledConnector`
6. Implement `showAngularHierarchy()` method in `HierarchyDisplayControl`.

Composition of Extensions



Task:

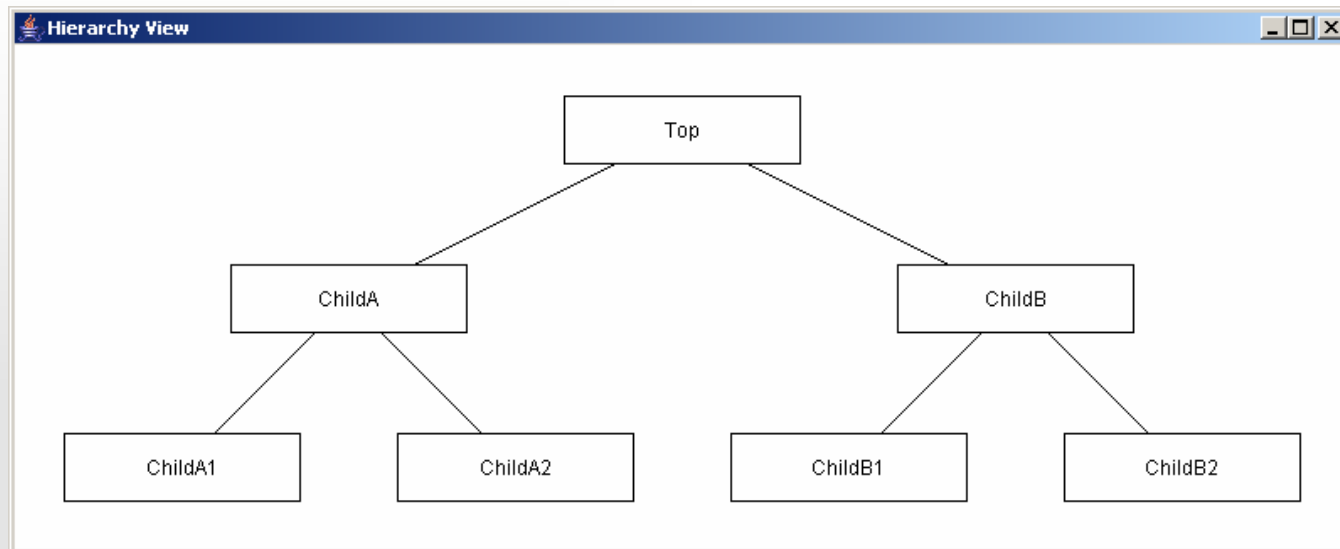
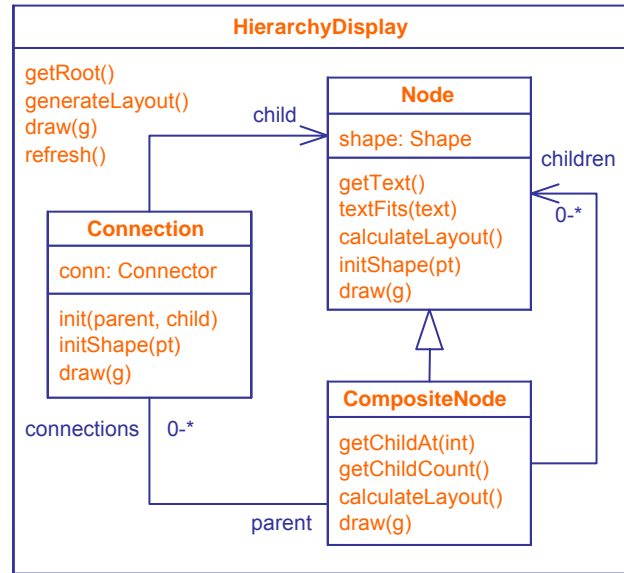
Create a hierarchy display component with both extensions: nodes adjusted to text size and angled connectors.

Steps:

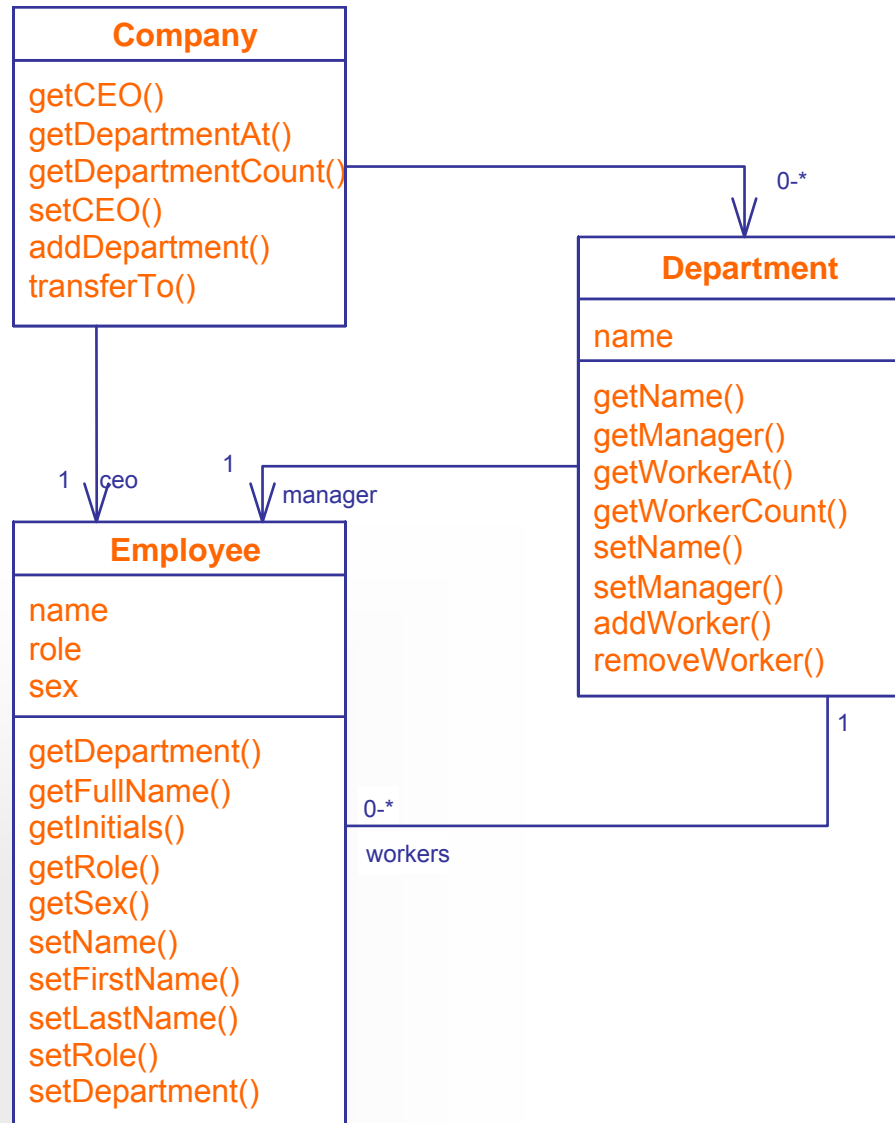
1. Declare `AdjustedAngularHierarchyDisplay` as combination of `AdjustedHierarchyDisplay` and `AngularHierarchyDisplay`
2. Implement `showAdjustedAngularHierarchy()` method in `HierarchyDisplayControl`.

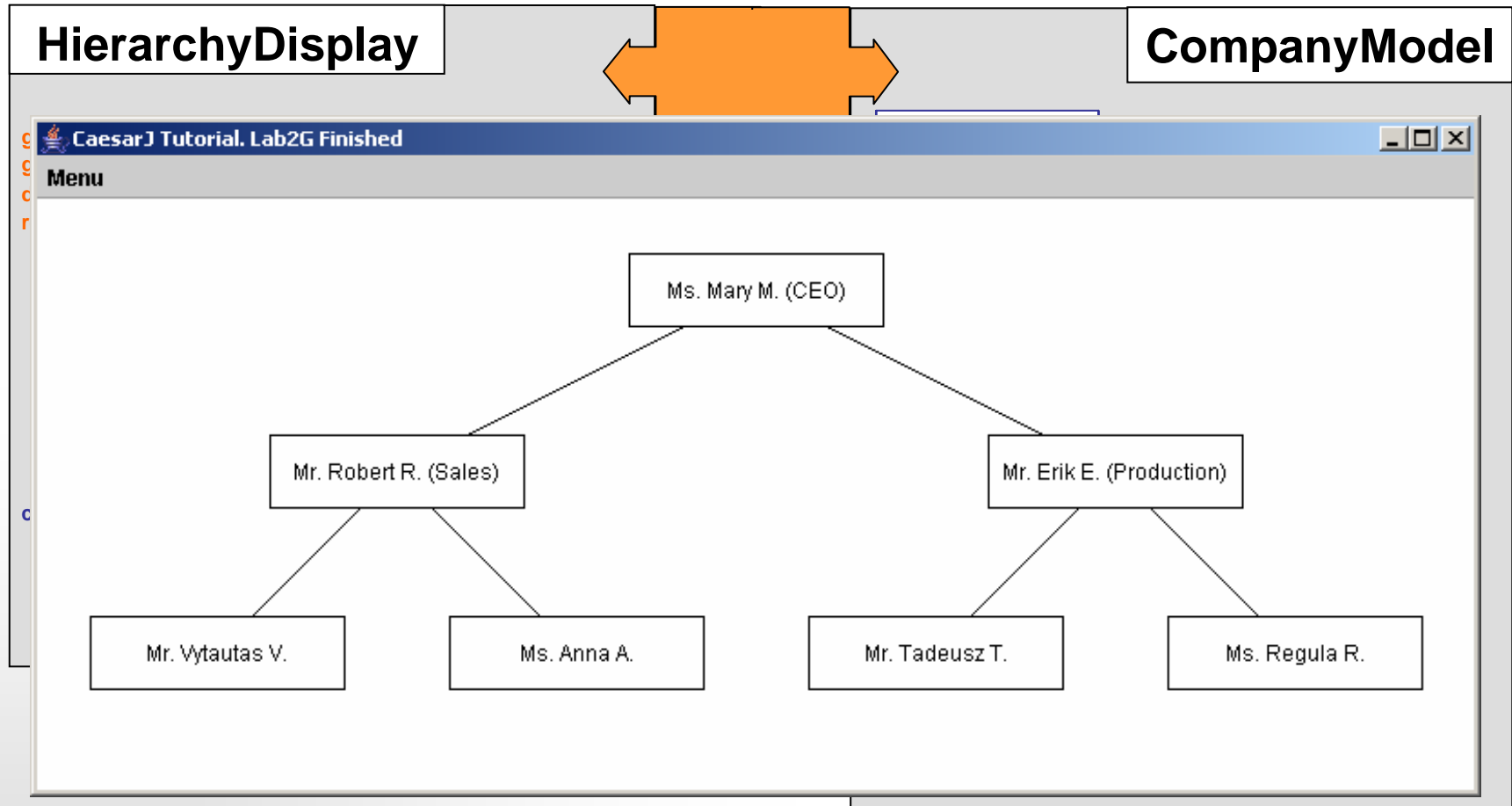
- Component Extension
 - Extending component with virtual classes
 - Combining different extensions
- **Component Integration**
 - Defining wrapper classes
 - Observing events with pointcuts
 - Dynamic aspect deployment
- Variability Management
 - Varying component implementations
 - Varying component bindings
 - Extending collaboration interface
- Integrating Distributed Components

Component A: Hierarchy Display



Application Data Model

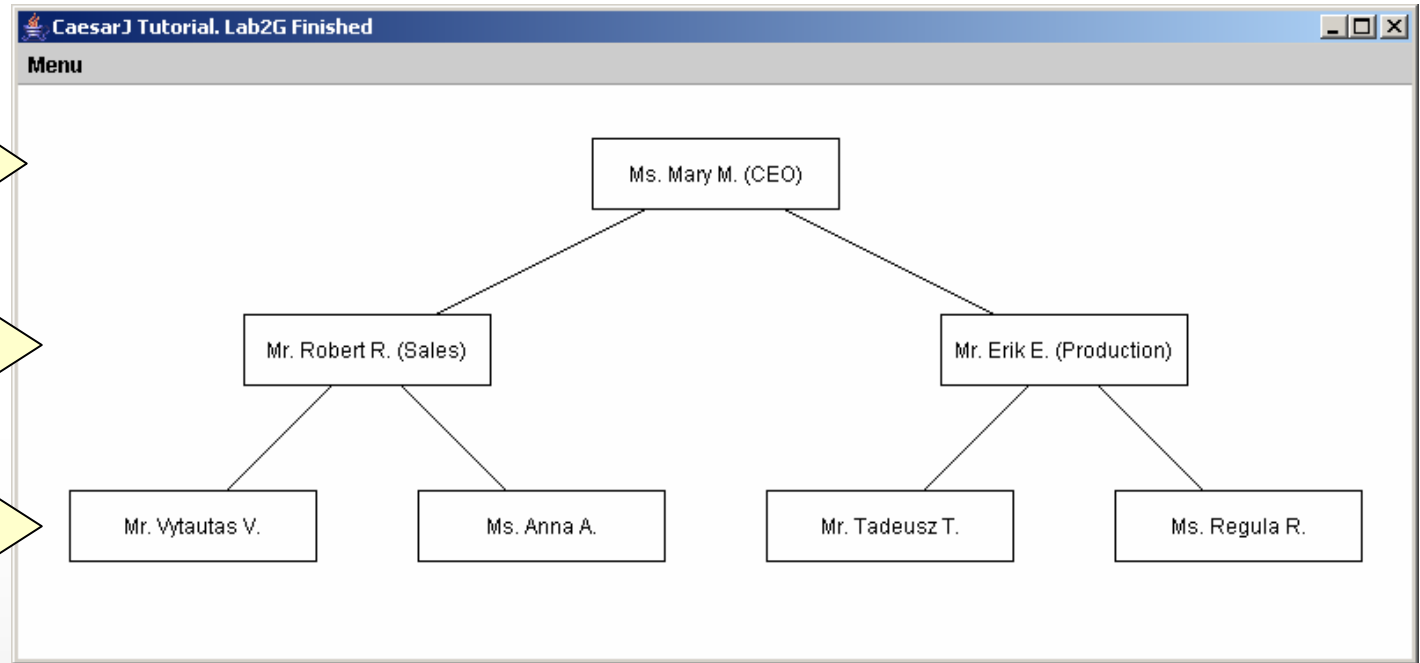




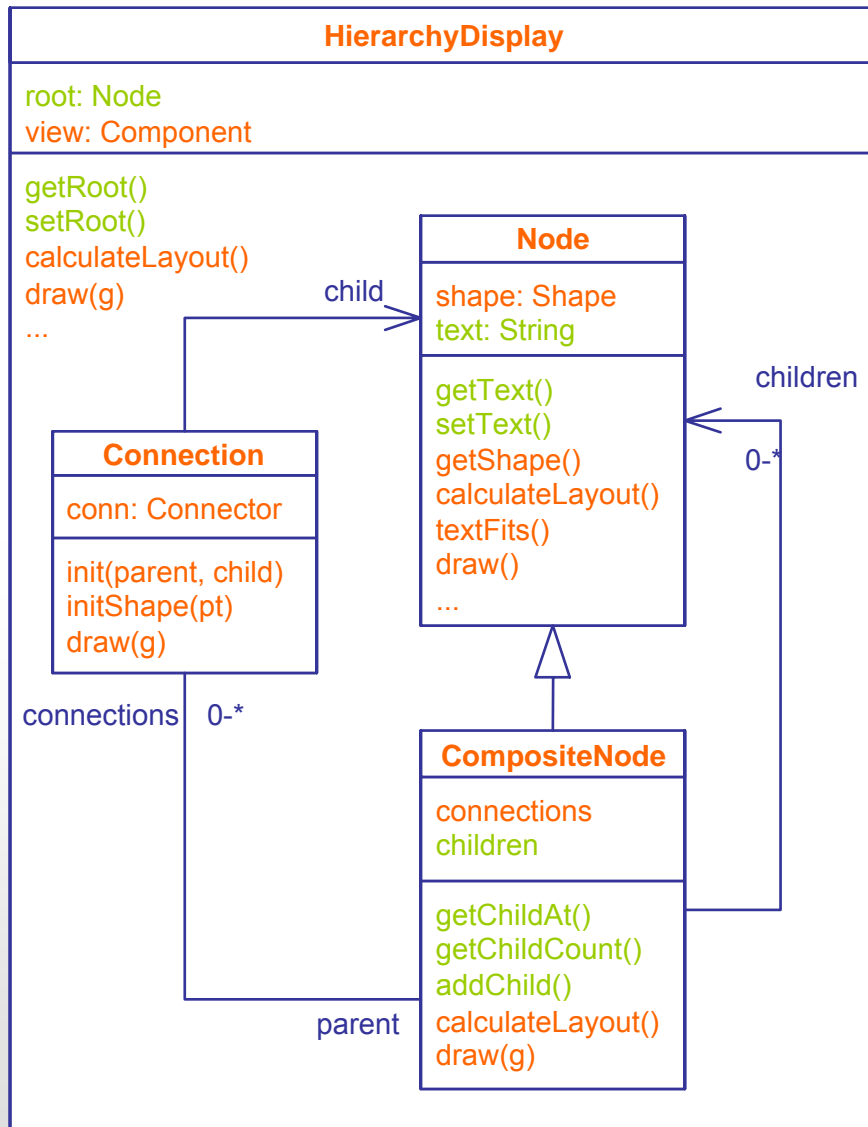
CEO

Department Managers

Workers



Refactoring: Separation of Concerns



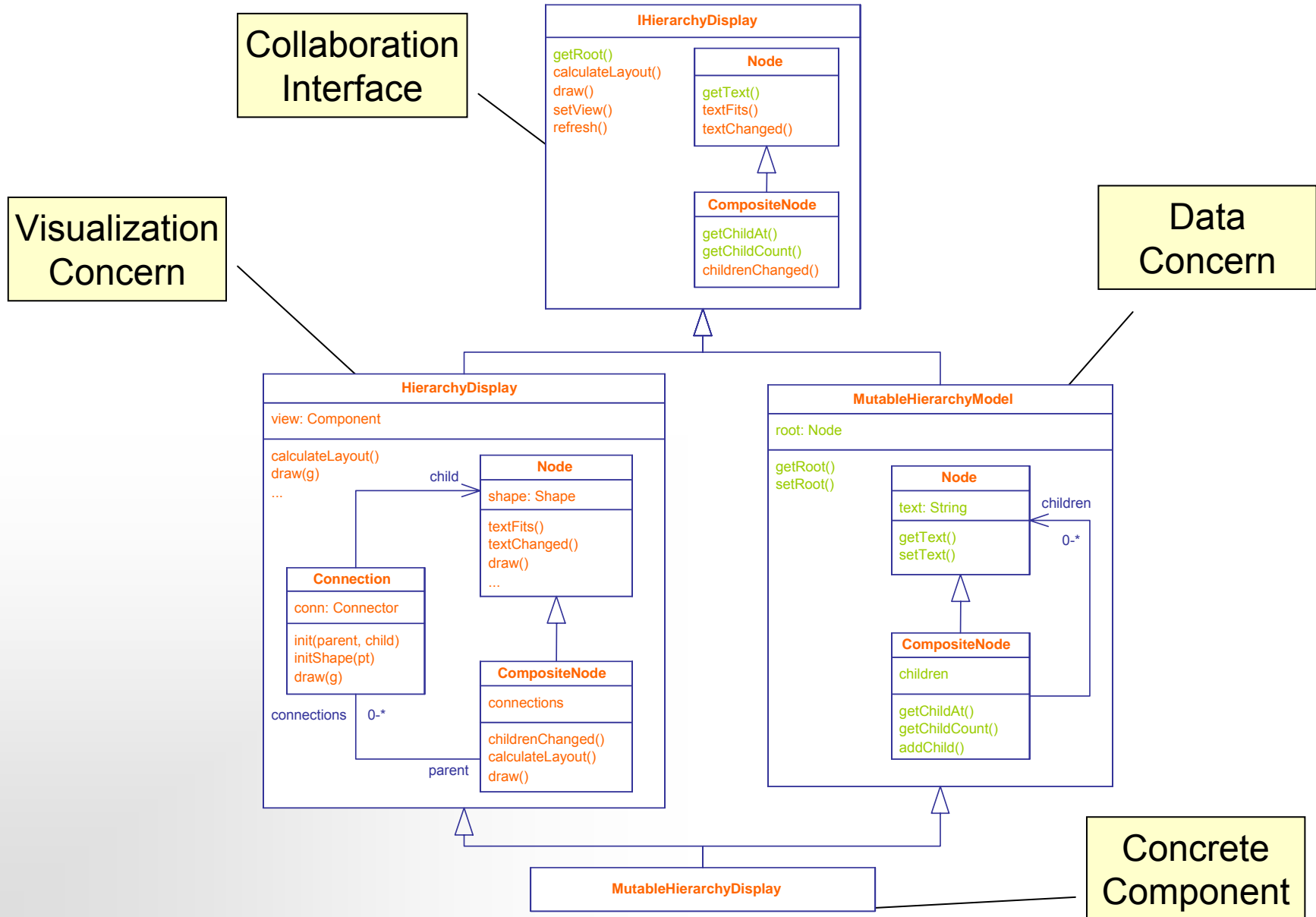
Tightly Coupled Concerns:

- Data model
- Visualization

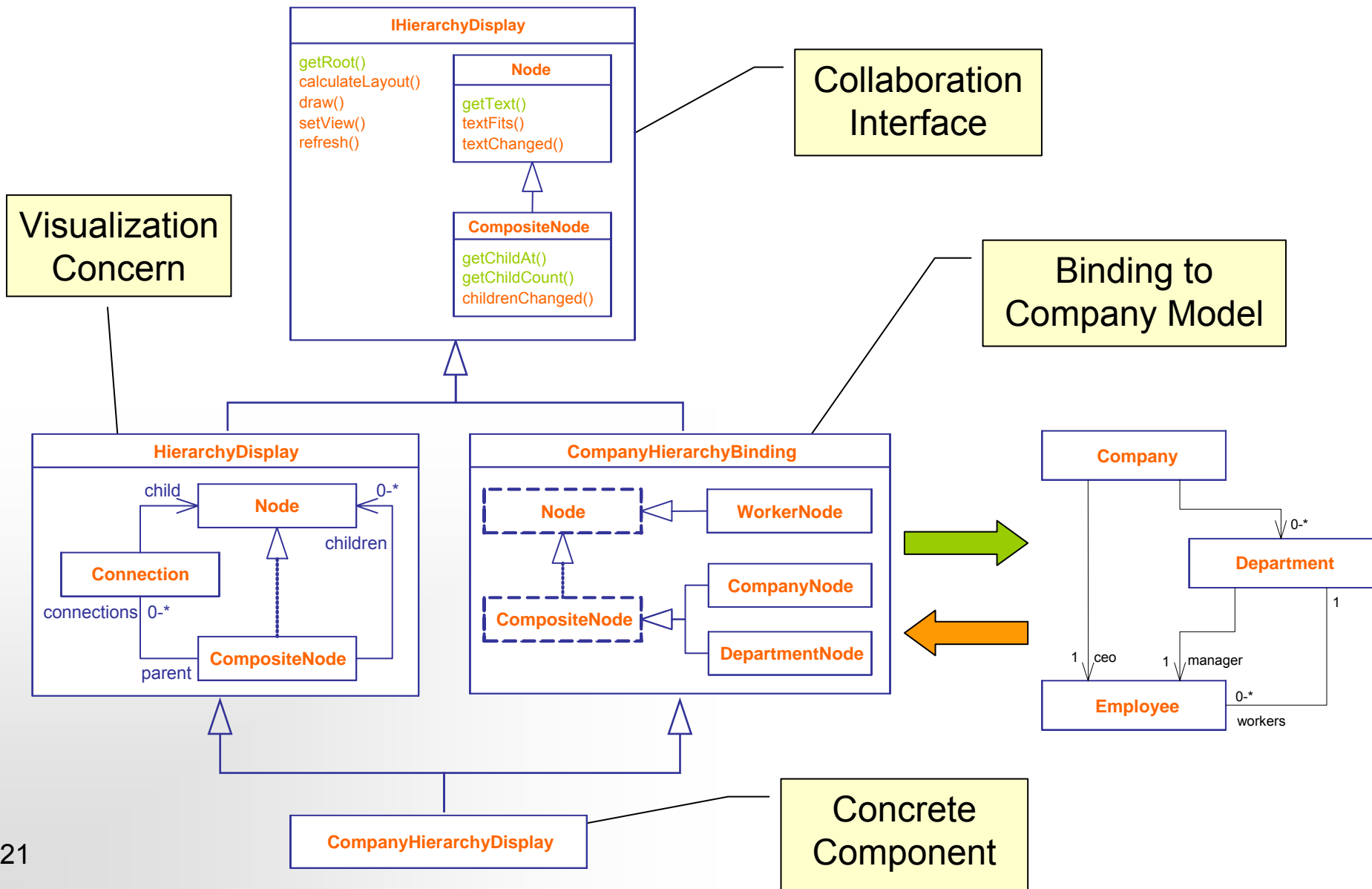
Design Goal:

- Make the hierarchy display reusable with various data models

Refactoring: Separation of Concerns



Component Integration



←caesarj→ Wrapper Recycling

```
cclass CompanyHierarchyBinding {  
  cclass WorkerNode  
    extends Node wraps Employee { ... }  
  ...  
}
```

```
void test() {  
  CompanyHierarchyBinding hier =  
    new CompanyHierarchyDisplay();  
  
  Employee anna = new Employee();  
  
  assert(hier.WorkerNode(anna) ==  
         hier.WorkerNode(anna));  
  
  Employee peter = new Employee();  
  
  assert(hier.WorkerNode(anna) !=  
         hier.WorkerNode(peter));  
}
```

wrapper is created on demand

wrapper is reused for the same object

wrapper is destroyed by garbage collector

<caesarj> To Do: Extend Company Hierarchy

Task:

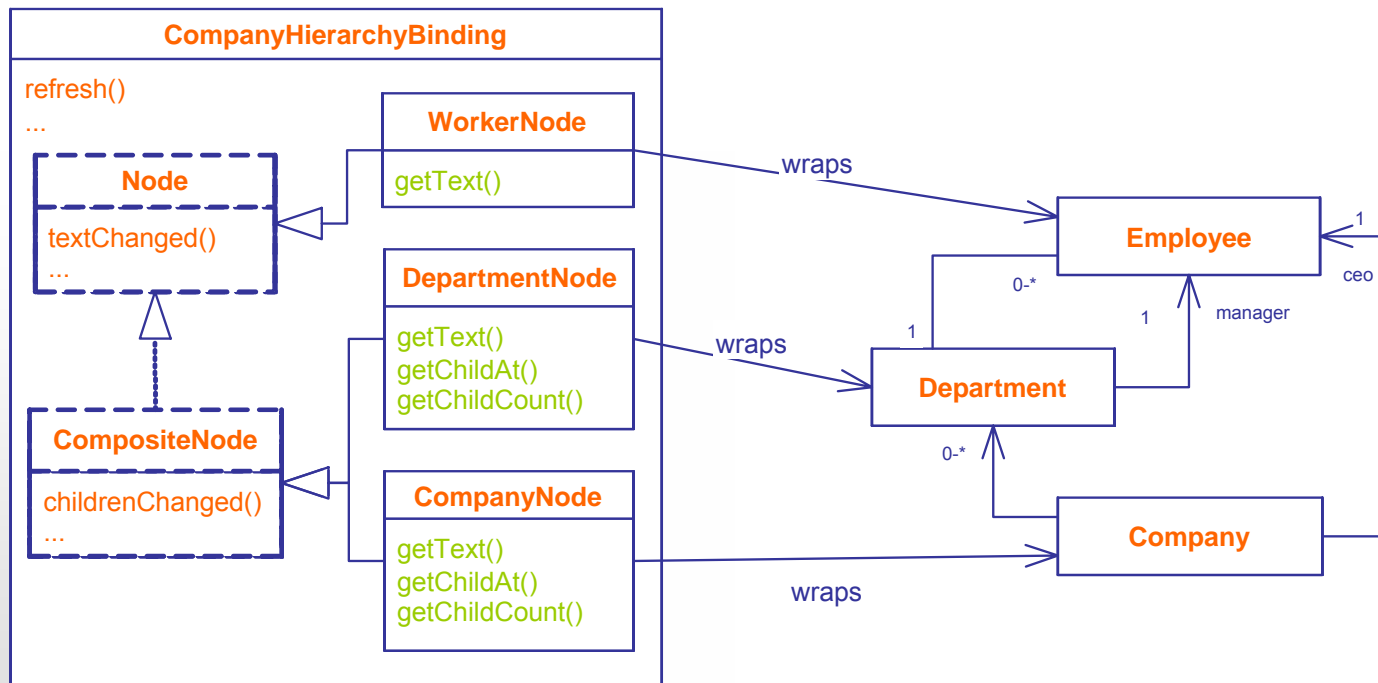
Extend company hierarchy binding with nodes to display department workers

Steps:

1. Open **Lab2B**
2. Declare **WorkerNode** wrapper class
3. Implement **getText()** using methods of **wrappee**
4. Declare **DepartmentNode** as subtype of **CompositeNode**
5. Implement **getChildAt()** and **getChildCount()** for **DepartmentNode**

Component Integration - Pointcuts

After certain changes display must be updated



<caesarj> Aspects in CaesarJ

AspectJ aspects:

```
public aspect MyAspectClass {  
    private String id = "singleton";  
    public void setId(String id) {  
        this.id = id;  
    }  
    after() : execution(* Foo.foo(..)) {  
        System.out.println("foo " + id);  
    }  
}
```

AspectJ aspects are statically
deployed classes of CaesarJ



```
deployed public cclass MyAspectClass {  
    private String id = "singleton";  
    public void setId(String id) {  
        this.id = id;  
    }  
    after() : execution(* Foo.foo(..)) {  
        System.out.println("foo " + id);  
    }  
}
```

<caesarj> Dynamic Aspect Deployment

But aspects can also be deployed dynamically

```
MyAspectClass asp1 = new MyAspectClass();
MyAspectClass asp2 = new MyAspectClass();

new Foo().foo(); /* not intercepted */
deploy asp1;
new Foo().foo(); /* intercepted by asp1 */
deploy asp2;
new Foo().foo(); /* intercepted by asp1 and asp2 */
undeploy asp1;
undeploy asp2;
new Foo().foo(); /* not intercepted */
```

<caesarj> Aspect Scoping

Tread local deployment:

```
MyAspectClass asp1 = new MyAspectClass();
DeploySupport.deployOnThread(asp1);
new Foo().foo();
DeploySupport.undeployFromThread(asp1);
```

Object local deployment:

```
MyAspectClass asp1 = new MyAspectClass();
MyAspectClass asp2 = new MyAspectClass();
Foo foo1 = new Foo();
Foo foo2 = new Foo();
DeploySupport.deployOnObject(asp1, foo1);
DeploySupport.deployOnObject(asp2, foo2);
foo1.foo();           /* intercepted by asp1 */
foo2.foo();           /* intercepted by asp2 */
new Foo().foo();      /* not intercepted */
DeploySupport.undeployFromObject(asp1, foo1);
DeploySupport.undeployFromObject(asp2, foo2);
```

<caesarj> To Do: Deploy Display Components

Task:

Deploy hierarchy display components when they are used, undeploy them when they are not used.

Steps:

- Open **Lab2C**
- Extend **switchCompanyHierarchy()** in **HierarchyDisplayControl**

– Undeploy old hierarchy display component

```
undeploy view.getHierarchy();
```

– Deploy new hierarchy display component

```
deploy hier;
```

Component Integration - Pointcuts

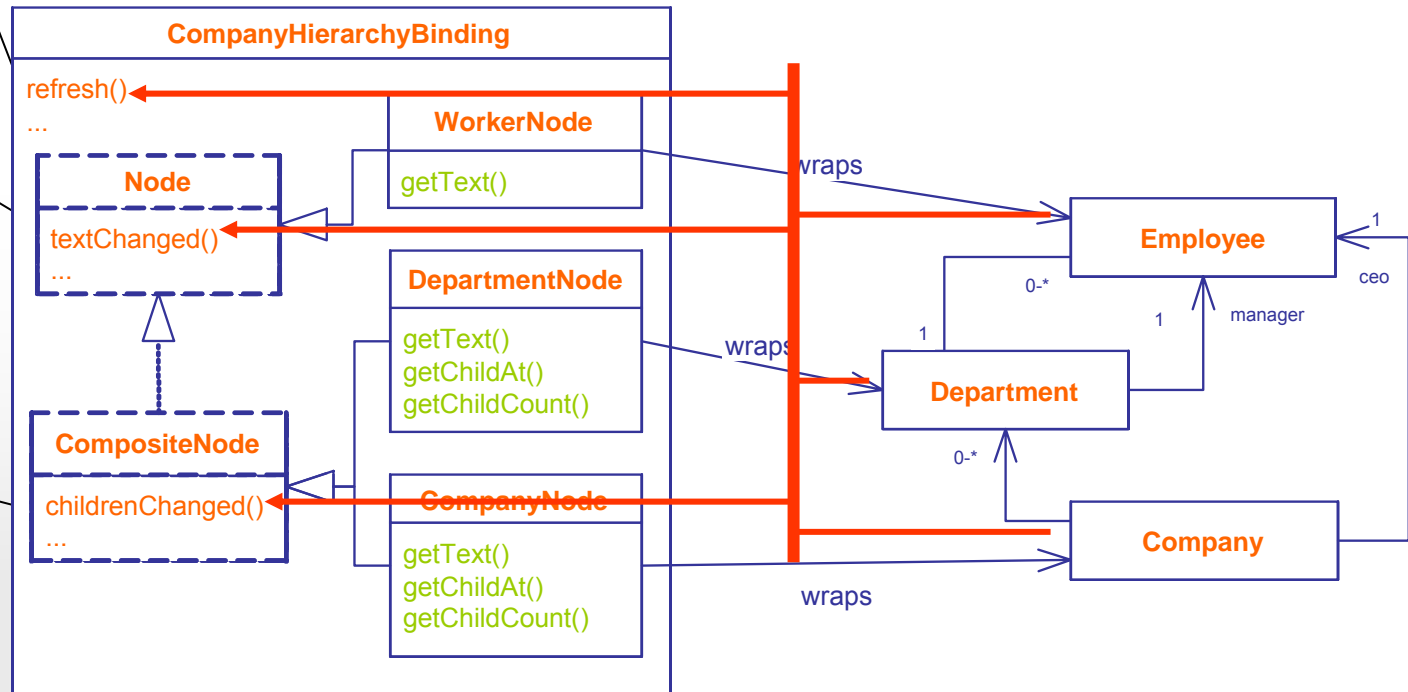
After certain changes display must be updated



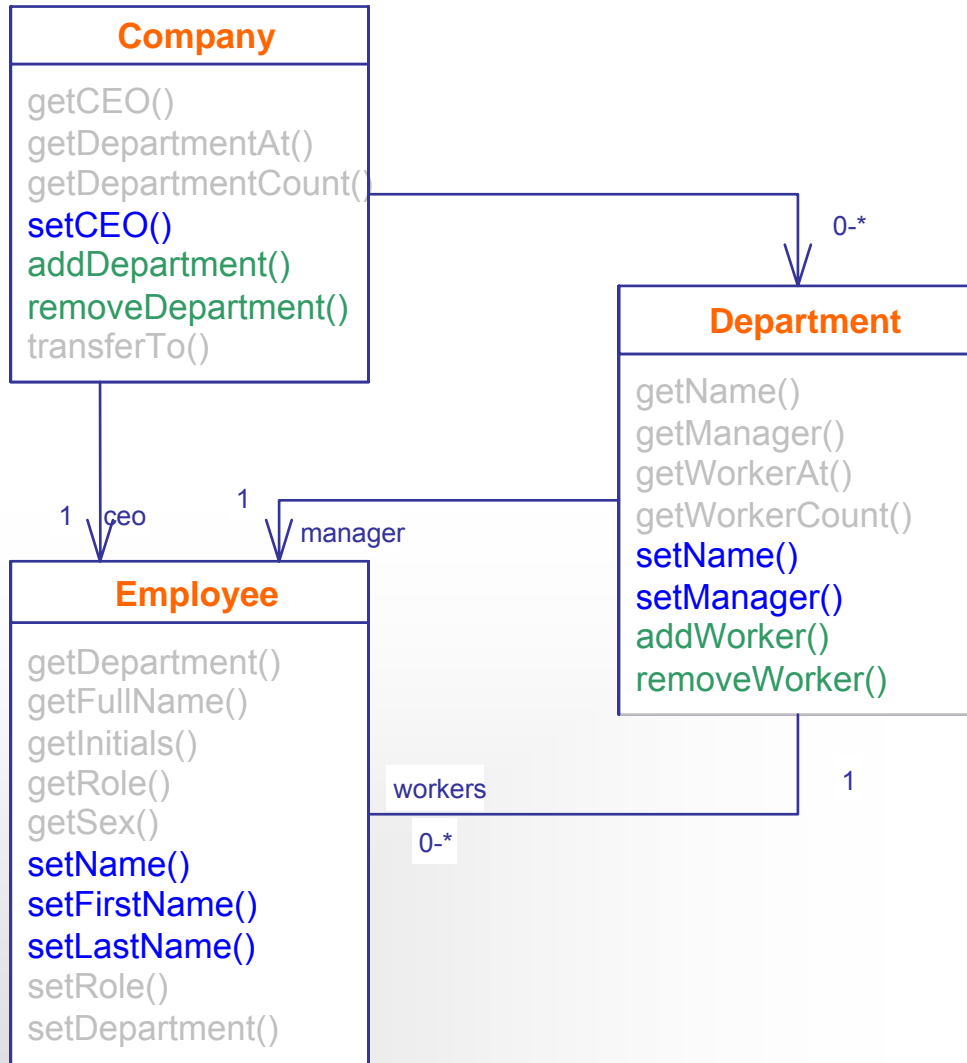
Call when redraw is needed

Call after text change

Call after children changed



Changes in Company Model



Affects text of a node



Affects children of a node



No direct effect

Task:

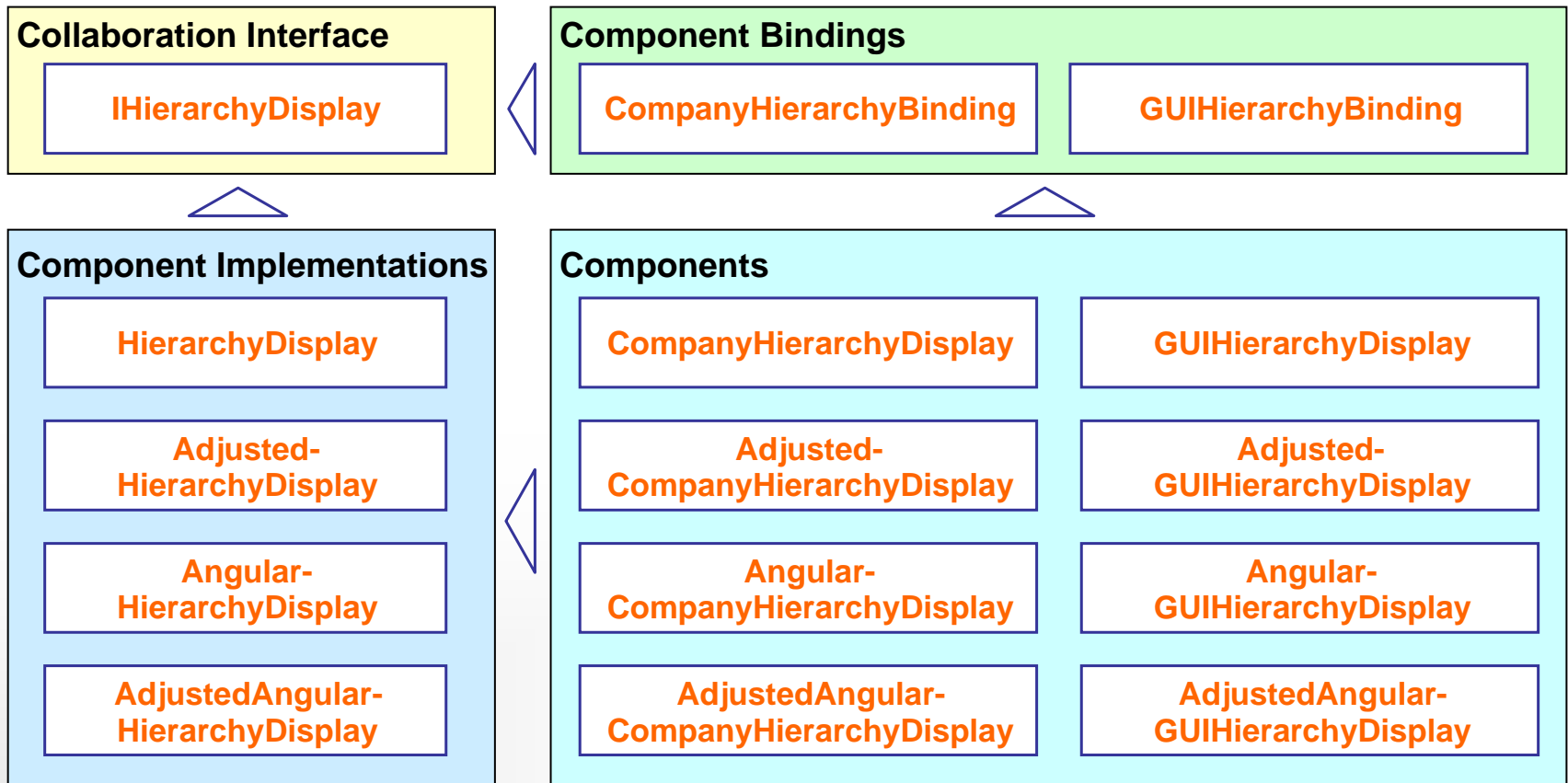
Complete pointcuts and advice to observe text changes of worker nodes and children changes of department nodes.

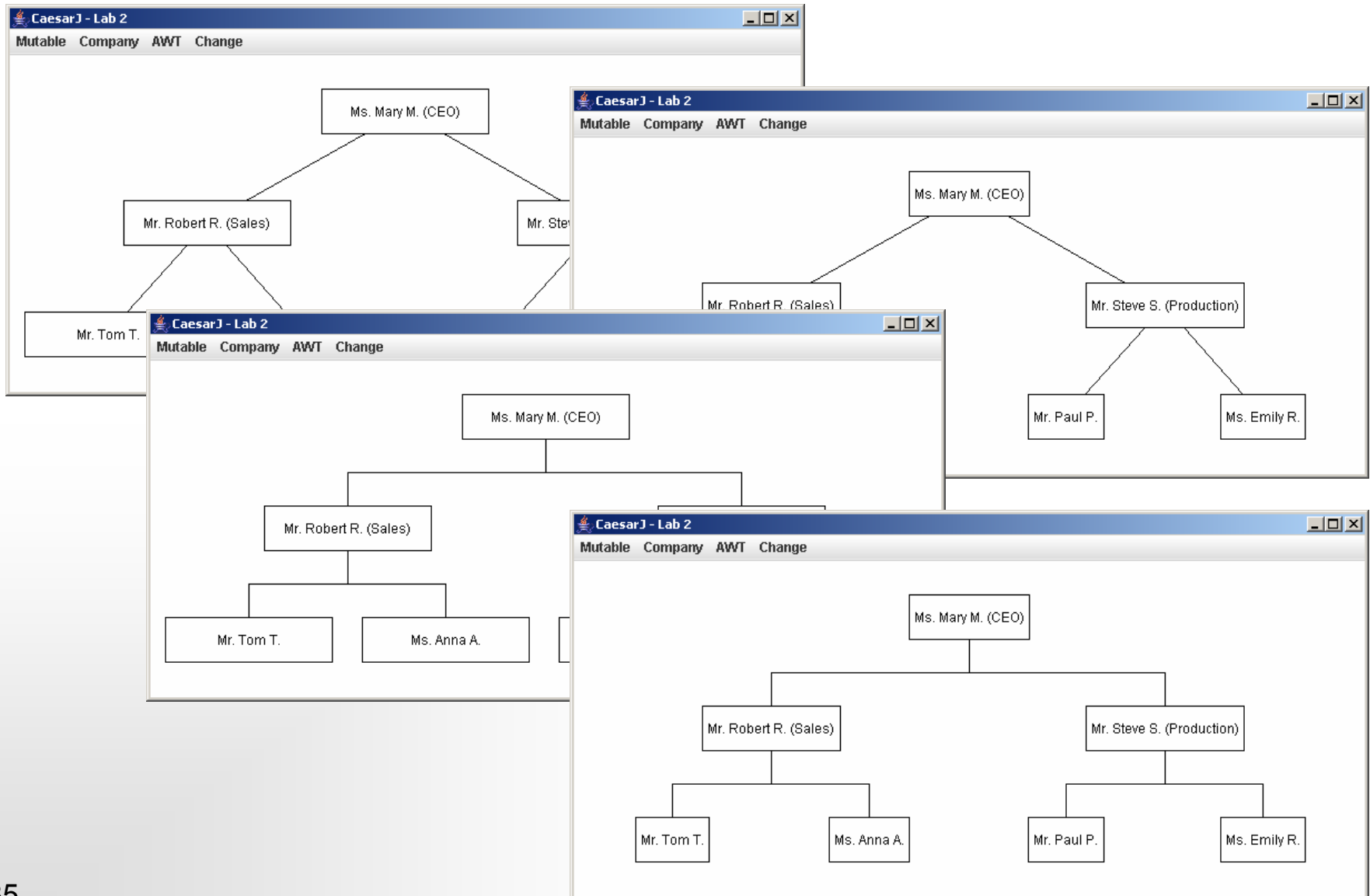
Steps:

- In `CompanyHierarchyBinding` extend advice after Employee name change to update corresponding `WorkerNode`
- Write new advice with pointcut to observe changes of `DepartmentNode` children

- Component Extension
 - Extending component with virtual classes
 - Combining different extensions
- Component Integration
 - Defining wrapper classes
 - Observing events with pointcuts
 - Dynamic aspect deployment
- **Variability Management**
 - Varying component implementations
 - Varying component bindings
 - Extending collaboration interface
- Integrating Distributed Components

Combining Variations





<caesarj> To Do: Vary Display Implementations

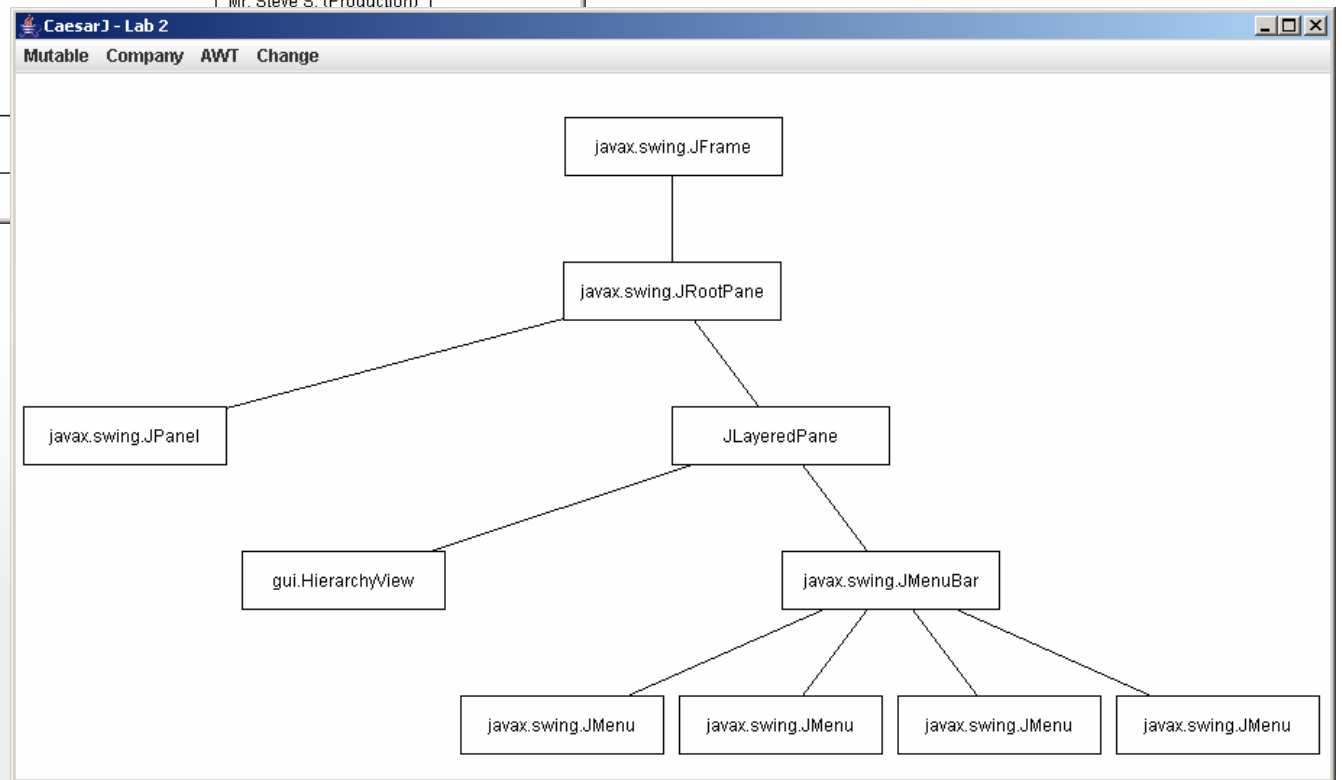
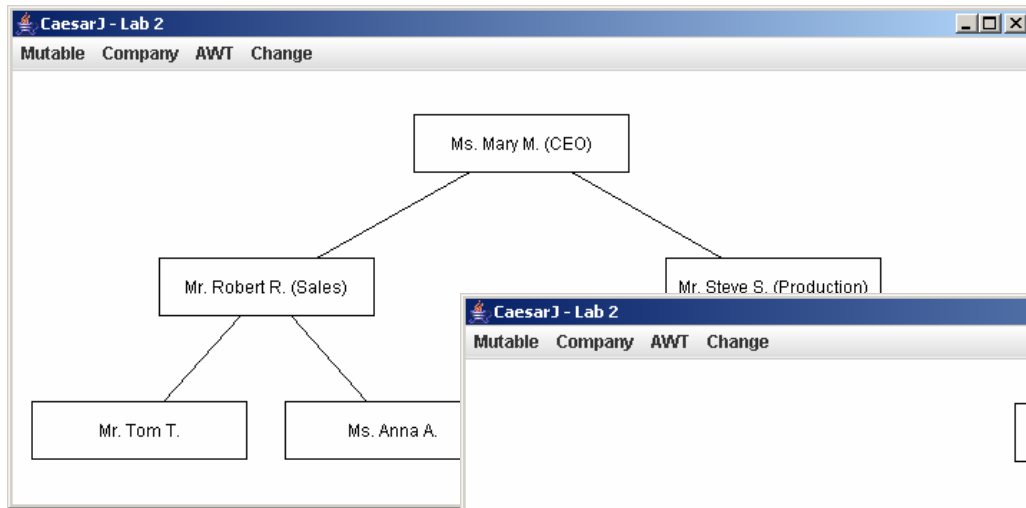
Task:

Combine company hierarchy binding with different hierarchy display implementations

Steps:

1. Open **Lab2D**
2. Define new classes as combinations of **CompanyHierarchyBinding** with various refinements of **HierarchyDisplay**
3. Implement **showXXXCompanyHierarchy()** methods in **HierarchyDisplayControl**

Component Binding Variations



<caesarj> To Do: Vary Component Bindings

Task:

Implement components to display window hierarchy of the application

Steps:

1. Combine `GUIHierarchyBinding` binding with different `HierarchyDisplay` implementations.
2. Implement `showXXXGUIHierarchy()` methods in `HierarchyDisplayControl`

- Component Extension
 - Extending component with virtual classes
 - Combining different extensions
- Component Integration
 - Defining wrapper classes
 - Observing events with pointcuts
 - Dynamic aspect deployment
- Variability Management
 - Varying component implementations
 - Varying component bindings
 - Extending collaboration interface
- **Integrating Distributed Components**

<caesarj> Remote Caesar Classes

Transparent usage:

```
cclass Company {
    public Employee getCEO() {
        return ceo;
    }
    ...
}

CaesarHost host = new CaesarHost("rmi://localhost/");
Company company = (Company)host.resolve("Company");
String ceoName = company.getCEO().getName();
...
```

but

- Do not compare remote references, implement **equals()** instead
- Do not forget to export all remote objects
- Prepare remote classes with RMI compiler
- Handle **CaesarRemoteException**

Caesar RMI Compiler

```
<target name="rmic" depends="compile">
  <java classname="org.caesarj.rmi.Compiler">
    <classpath refid="caesar.classpath"/>
    <arg value="-d"/>
    <arg value="\${destdir}"/>
    <arg value="-c"/>
    <arg value="\${projrmiclasspath}"/>
    <arg value="-r"/>
    <arg value="company.Company"/>
    <arg value="company.Department"/>
    <arg value="company.Employee"/>
    <arg value="hierarchies.company.CompanyHierarchyDisplay"/>
    <arg value="hierarchies.company.AdjustedCompanyHierarchyDisplay"/>
    <arg value="hierarchies.company.AngularCompanyHierarchyDisplay"/>
    <arg value="hierarchies.company.AdjustedAngularCompanyHierarchyDisplay"/>
  </java>
</target>
```

- Prepares Caesar classes
- Use option **-r** of to prepare inner classes
- Use standard RMI compiler for Java classes

<caesarj> Remote Aspect Deployment

Server side:

```
CaesarHost host = new CaesarHost("rmi://localhost/");
host.activateAspectDeployment();
```

Client side:

```
CaesarHost host = new CaesarHost("rmi://localhost/");
ICompanyHierarchyDisplay hier
    = new CompanyHierarchyDisplay();
host.deployAspect(hier);
...
host.undeployAspect(hier);
```

- Use **execution** pointcut, because calls can cross process boundaries
- Pointcut **cflow** cannot cross thread boundaries
- Prepare aspect classes with Caesar RMI compiler

Task:

Use different colors to display different types of company hierarchy nodes.

Steps:

- Open **Lab2F** project
- Change deployment strategy of hierarchy objects to remote deployment
- Enable aspect deployment on server side
- Build project using **build.xml** script
- Run **ServerMain**
- Run **Application**

Thank you!



<http://caesarj.org>