

Programmazione 3
e
Laboratorio di Programmazione 3
Unified Modeling Language
e
Java

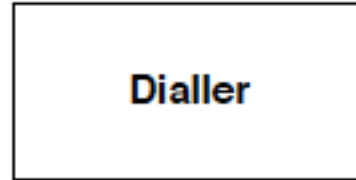
Angelo Ciaramella

Class Diagram

- Il **Class Diagram** permette di denotare il **contenuto statico** e le **relazioni tra classi**
- Il **Class Diagram** permette di **visualizzare relazioni e dipendenze strutturali** che **non sono riscontrabili scrivendo codice**



Classi



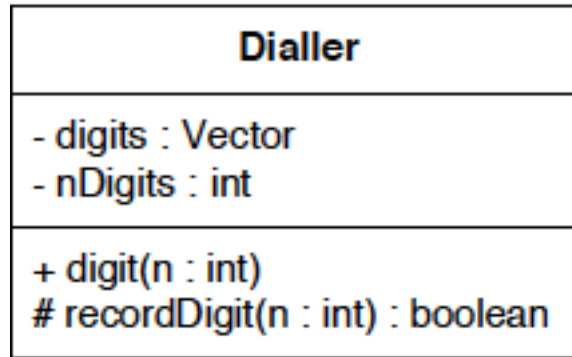
Rappresentazione di una classe in UML

```
public class Dialler
{
}
```

Rappresentazione in Java della classe



Classi



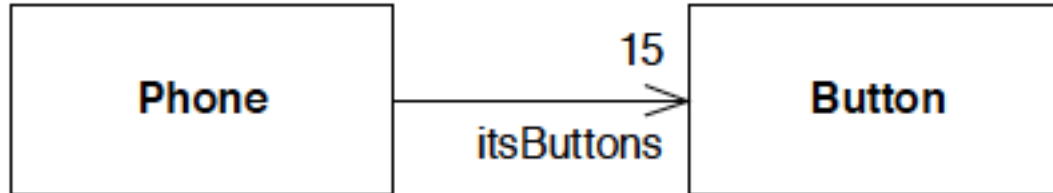
Dettagli della classe Dialler

```
public class Dialler
{
    private Vector digits;
    int nDigits;
    public void digit(int n);
    protected boolean recordDigit(int n);
}
```

Rappresentazione in Java della classe



Associazione



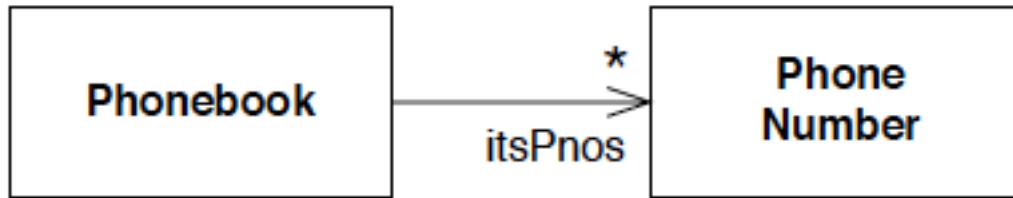
Associazione - la direzione della freccia ci dice che il telefono contiene un riferimento a pulsante

```
public class Phone
{
    private Button itsButtons[15];
}
```

Rappresentazione in Java della classe Phone



Molteplicità



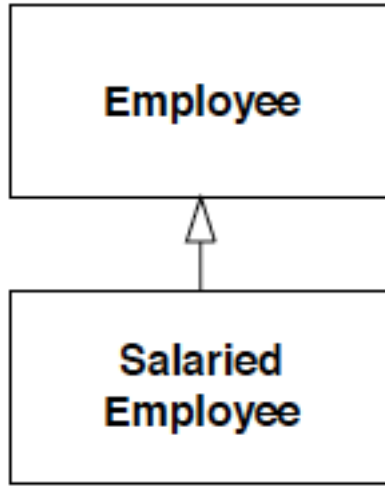
Molteplicità "many"

```
public class Phonebook
{
    private Vector itsPnos;
}
```

Rappresentazione in Java della classe Phonebook



Ereditarietà



Esempio di ereditarietà

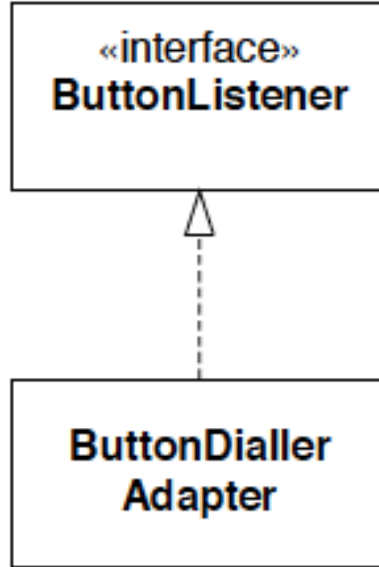
```
public class Employee
{
    ...
}

public class SalariedEmployee extends Employee
{
    ...
}
```

Rappresentazione in Java



Interfacce

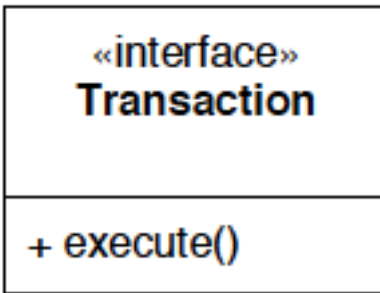


Implementazione di interfacce

```
interface ButtonListener
{
    ...
}

public class ButtonDiallerAdapter implements
ButtonListener
{
    ...
}
```


Stereotipi di classe



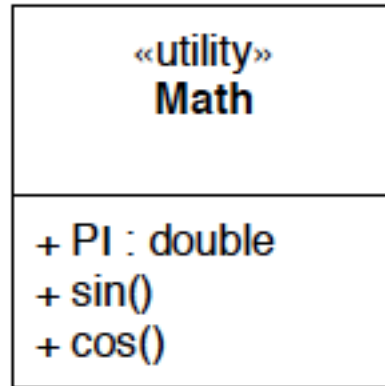
Stereotipo <<interface>>

```
interface Transaction
{
    public void execute();
}
```

Rappresentazione in Java



Stereotipi di classe



Stereotipo <<utility>>

```
public class Math
{
    public static final double PI = 3.14159265358979323;

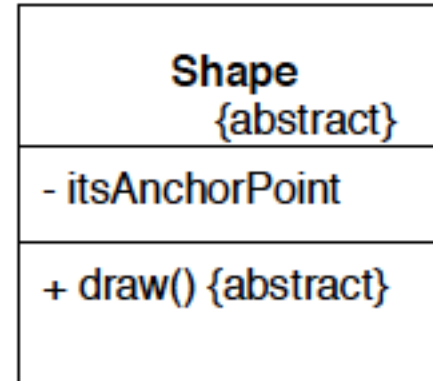
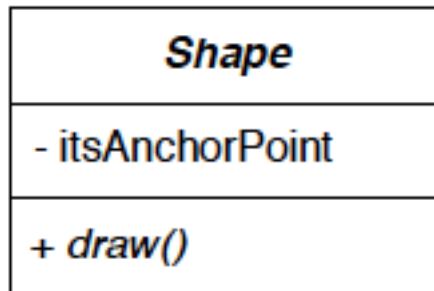
    public static double sin(double theta){...};

    public static double cos(double theta){...};
}
```

Rappresentazione in Java



Classi astratte



Rappresentazioni di classi astratte

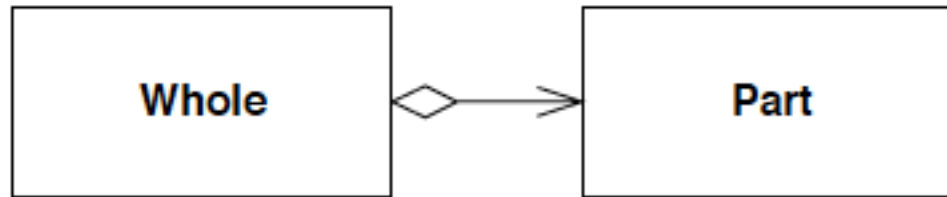
```
public abstract class Shape
{
    private Point itsAnchorPoint;

    public abstract void draw();
}
```

Rappresentazione in Java



Aggregazione



Rappresentazione di aggregazione (Parte-Intera)

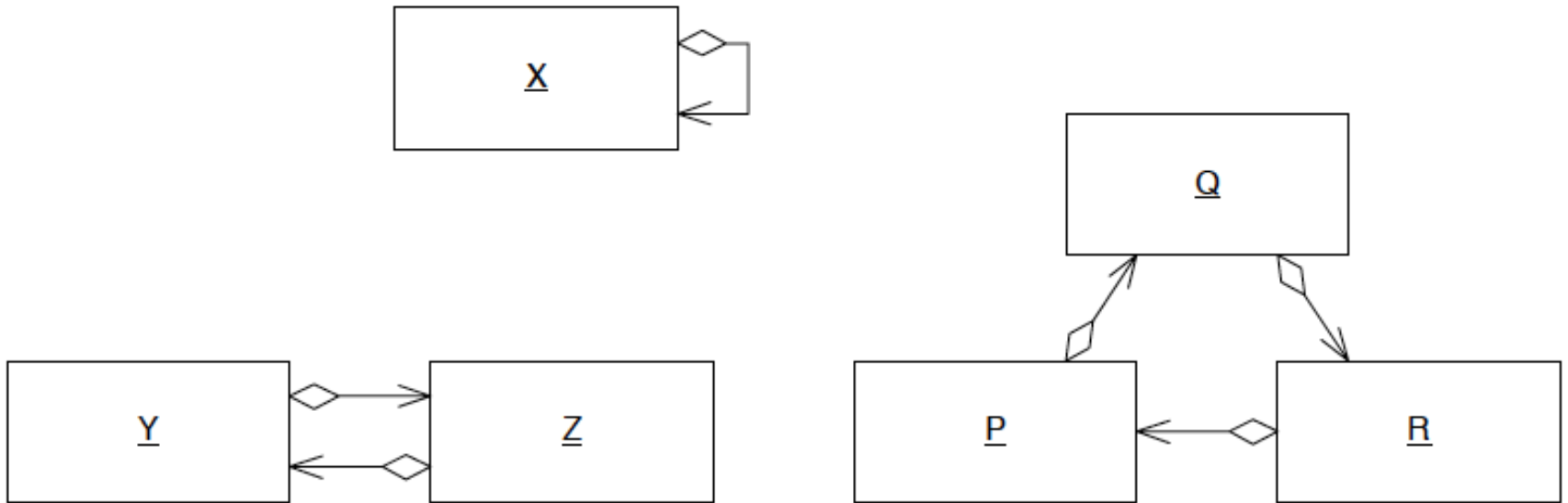
```
public class Whole
{
    private Part itsPart;
}
```

Rappresentazione in Java

Stesso risultato dell'aggregazione



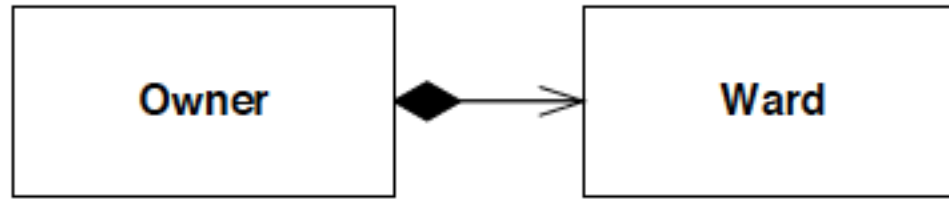
Aggregazione



Aggregazioni non ammesse tra istanze



Composizione



Rappresentazioni di composizione

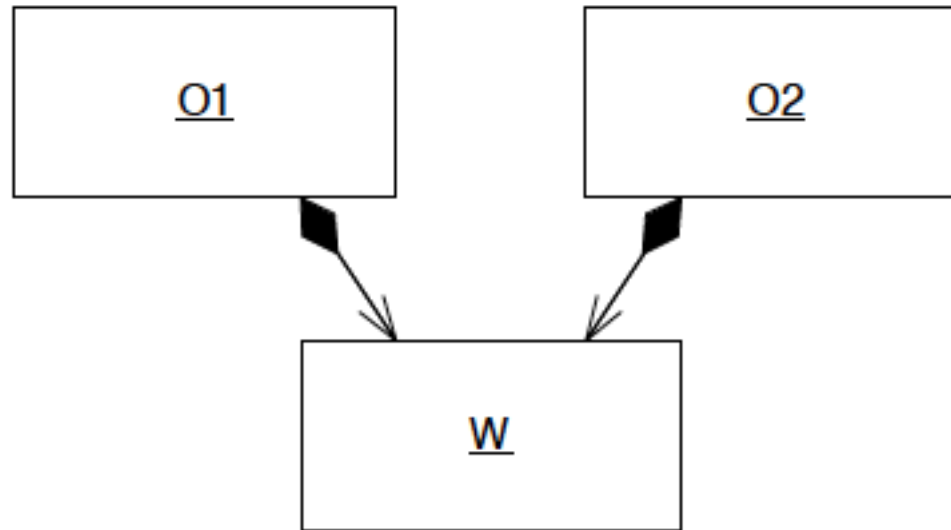
```
public class Owner
{
    private Ward itsWard;
}
```

Rappresentazione in Java

Stesso risultato dell'aggregazione



Composizione

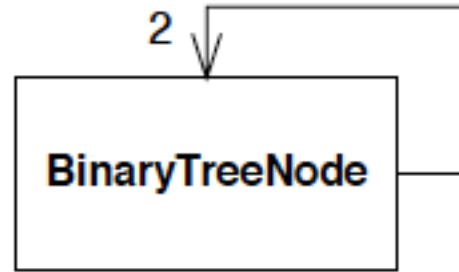


Composizione non ammessa tra istanze – ogni istanza delle classi componenti può appartenere a una sola istanza della classe composta

La classe composta è responsabile della durata della classe composta



Molteplicità

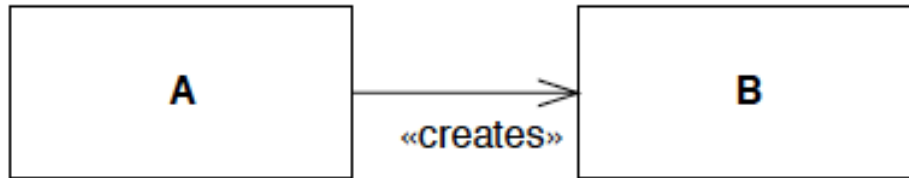


Semplice molteplicità

```
public class BinaryTreeNode
{
    private BinaryTreeNode leftNode;
    private BinaryTreeNode rightNode;
}
```

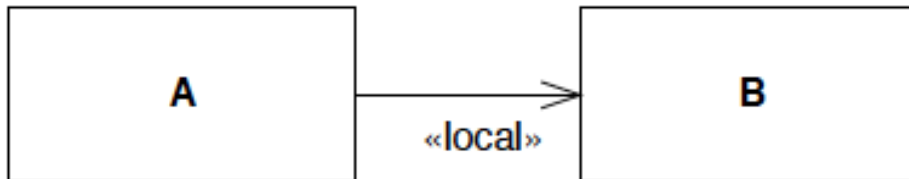


Stereotipi



Stereotipo

```
public class A
{
    public B makeB() {
        return new B();
    }
}
```

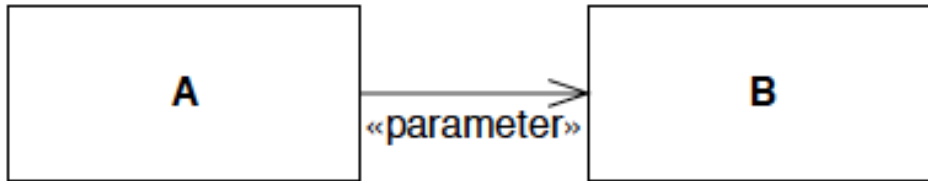


Stereotipo

```
public class A
{
    public void f() {
        B b = new B();
    }
}
```

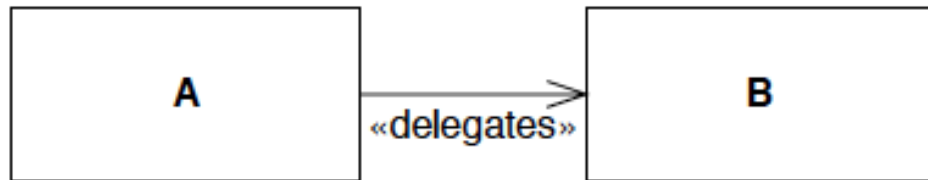


Stereotipi



Stereotipo

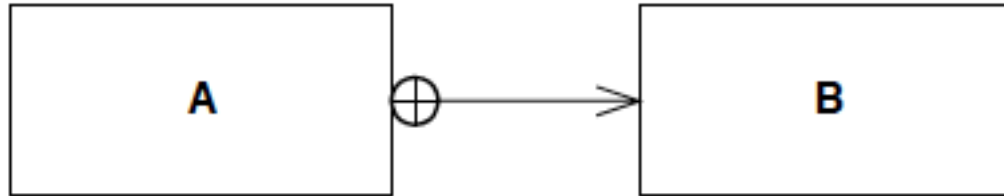
```
public class A
{
    public void f(b B) {
    }
}
```



Stereotipo

```
public class A
{
    private B itsB;
    public void f() {
        itsB.f();
    }
}
```

Classi nascoste

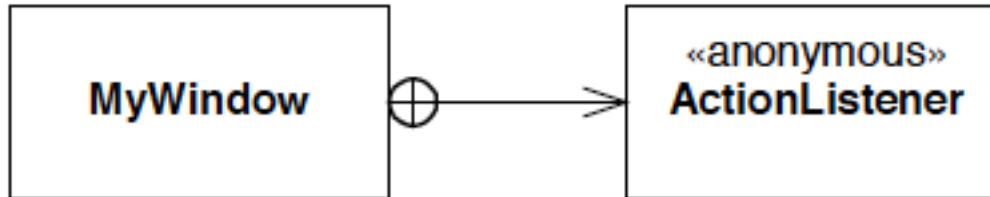


Classe nascosta

```
public class A {  
    private class B {  
        ...  
    }  
}
```



Classi nascoste

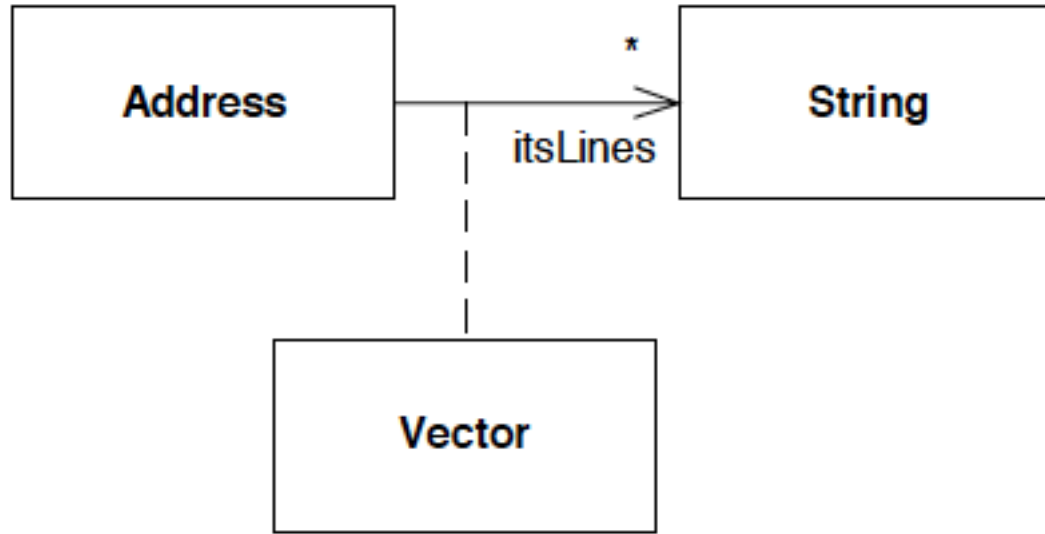


Classe nascosta anonima

```
public class Window {
    private void f() {
        ActionListener l = new ActionListener() {
            // implementazione
        };
    }
}
```



Association class



Association class

```
public class Address {  
    private Vector itsLines;  
};
```



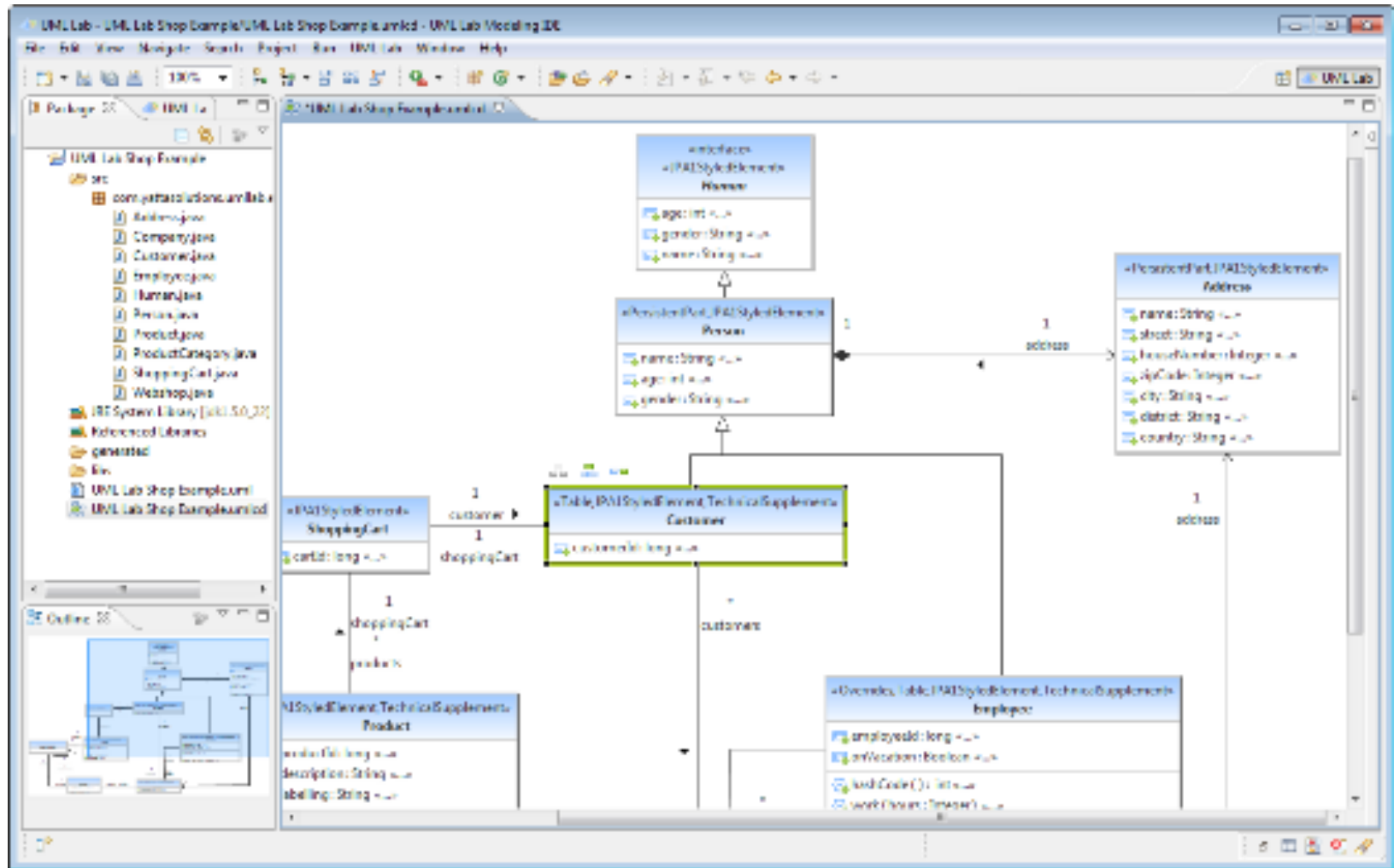
Software ... Eclipse



The screenshot displays the Eclipse IDE interface. The Package Explorer on the left shows the project structure for 'Jedi', with the 'jedi.functional' package selected. The main editor window shows the source code for 'FunctionalPrimitives.java'. The code includes a 'selected' list, a 'forEach' method, and a 'head' method. The Outline view on the right lists the methods in the 'FunctionalPrimitives' class, such as 'addToGroup', 'append', 'collect', 'drop', 'flatten', 'fold', 'forEach', 'group', 'head', 'headOrDefaultIfEmpty', 'join', 'listTabulate', 'longest', 'makeList', 'only', 'produce', 'random', 'randomOrder', 'reverse', 'select', 'sequence', 'shortest', 'slice', 'take', 'takeRight', and 'zip'.

```
121 List<T> selected = new ArrayList<T>();
122
123
124 for (T item : items) {
125     if (filter.execute(item)) {
126         selected.add(item);
127     }
128 }
129
130 return selected;
131 }
132
133 /**
134  * Iterate over a collection of <code>items</code> applying the given <code>command</code> to each
135  */
136 public static <T> Collection<T> forEach(Collection<T> items, Command<? super T> command) {
137     assertNotNull(command, "command");
138     assertNotNull(items, "items");
139
140     for (T item : items) {
141         command.execute(item);
142     }
143
144     return items;
145 }
146
147 /**
148  * Return the one and only item in the given collection.
149  *
150  * @return the item in the collection
151  * @throws JediAssertion.AssertionError if the collection contains less or more than one item
152  * @see #head(Collection)
153  * @see #headOrNullIfEmpty(Collection)
154  */
155 public static <T> T only(Collection<T> items) {
156     assertNotNull(items, "items");
157     assertEquals(1, items.size(), "only one item");
158     return head(items);
159 }
160
161 /**
162  * Get the first item (in iteration order) from a collection. The collection must contain at least one i
163  * {@link JediAssertion.AssertionError} will be thrown.
164  *
165  * @return the first item in the collection
166  * @throws JediAssertion.AssertionError if the collection contains less or more than one item
167  * @see #only(Collection)
168  * @see #headOrNullIfEmpty(Collection)
169  * @see #headOrDefaultIfEmpty(Collection, Object)
170  */
171 public static <T> T head(Collection<T> items) {
172     assertNotNull(items, "items");
173 }
```

Software ... UML Lab Class Diagrams

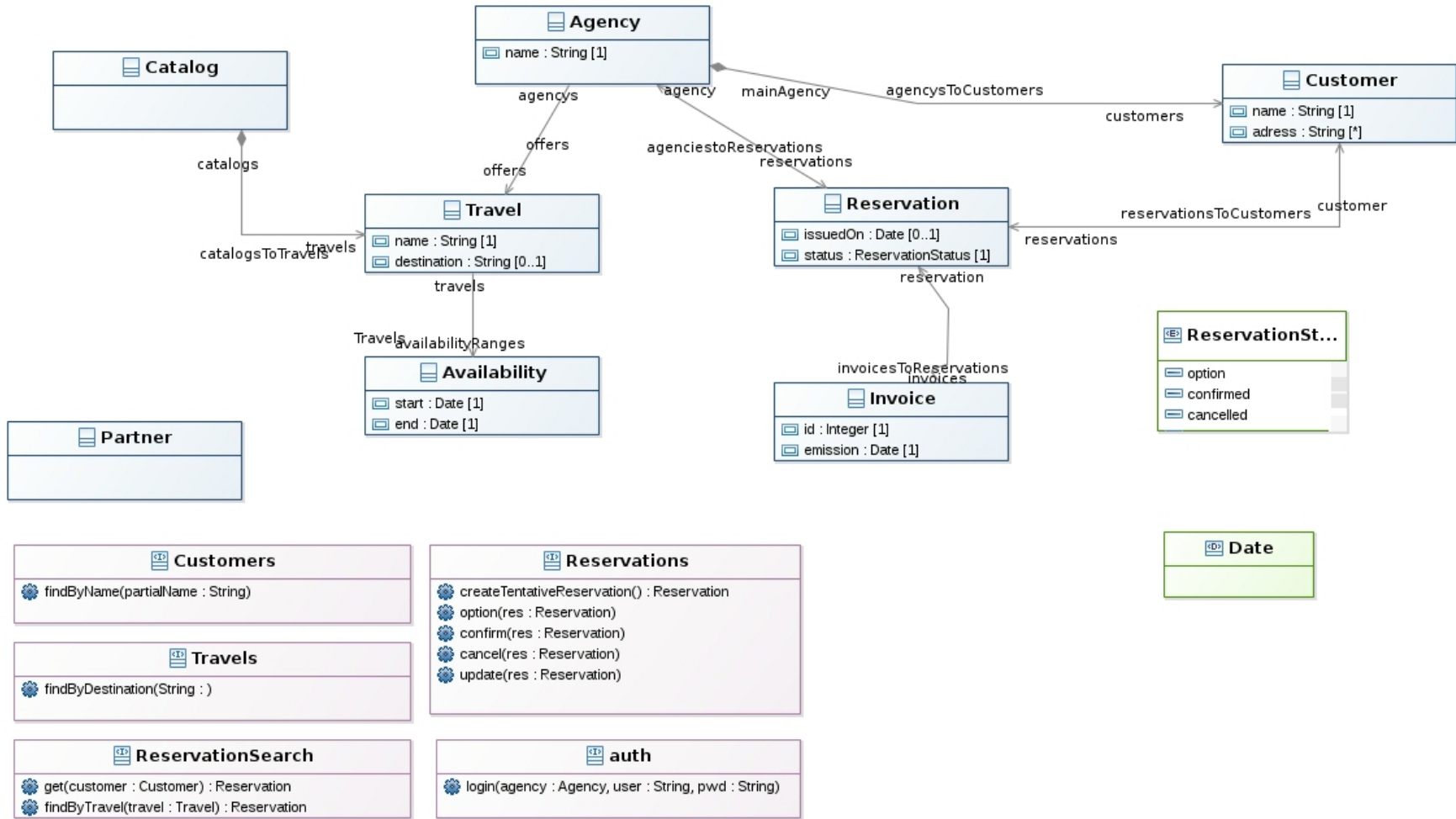


Software ... UML Designer

- Diagrammi di UML Designer
 - Package Hierarchy
 - Class Diagram
 - Component Diagram
 - Composite Structure Diagram
 - Deployment Diagram
 - Use Case Diagram
 - Activity Diagram
 - State Machine Sequence Diagram
 - Profile Diagram

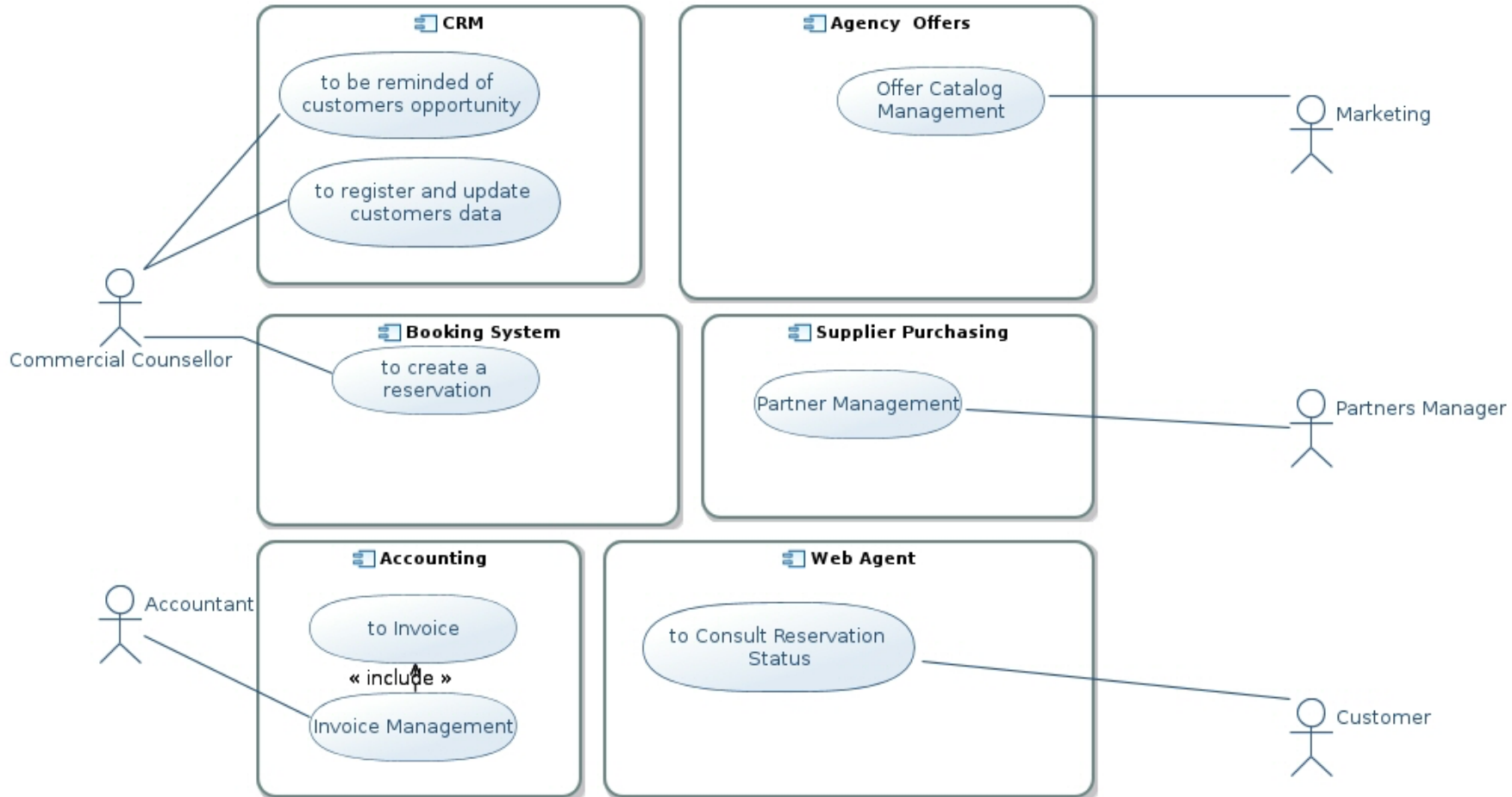


Software ...



Class Diagram

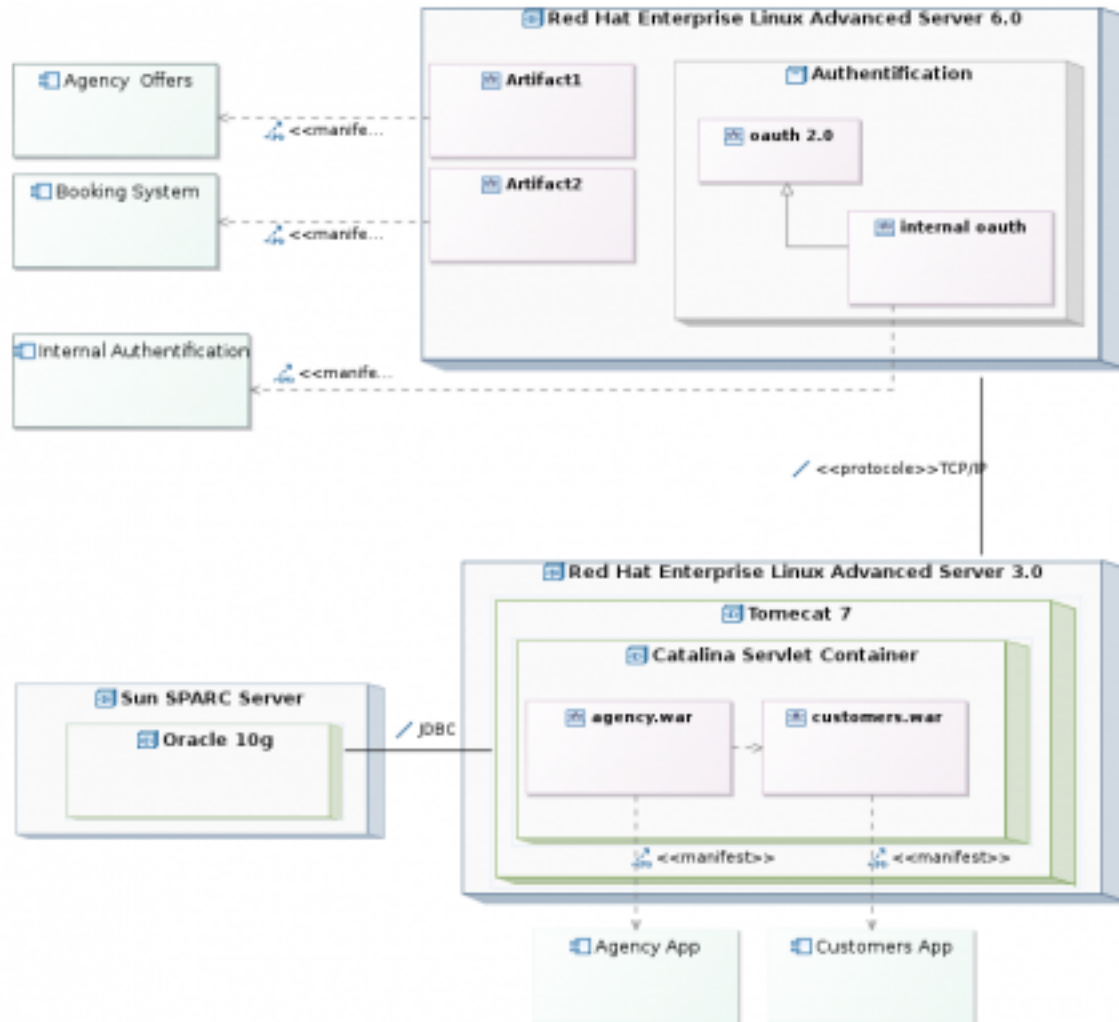
Software ...



Case Use Diagram

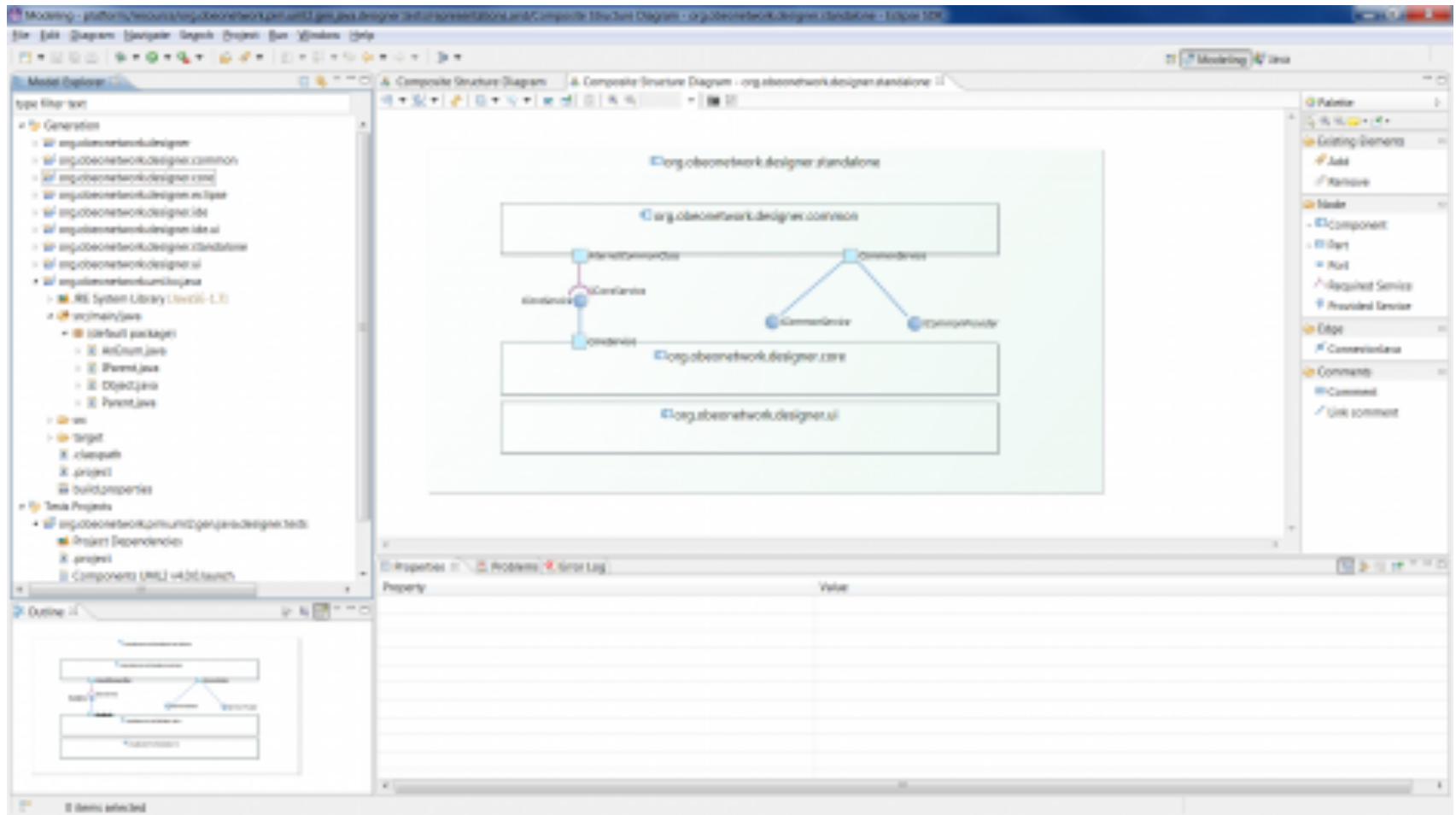


Software ...



Deployment Diagram

Software ... UML to Java Generator



Software ... UML to Java Generator

