

Programmazione 3

e

Laboratorio di Programmazione 3

Unified Modeling Language

e

Java

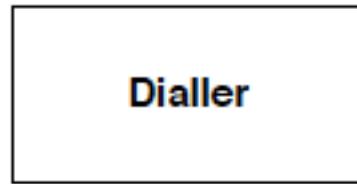
Angelo Ciaramella

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



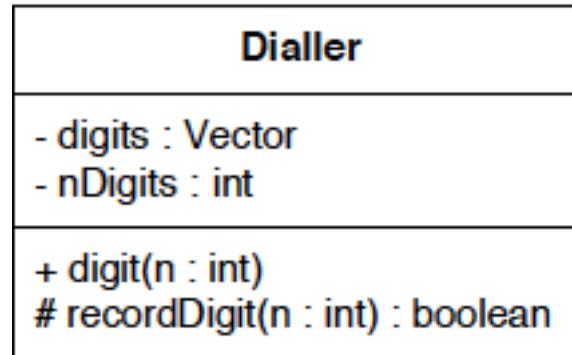
Rappresentazione di una classe in UML

```
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



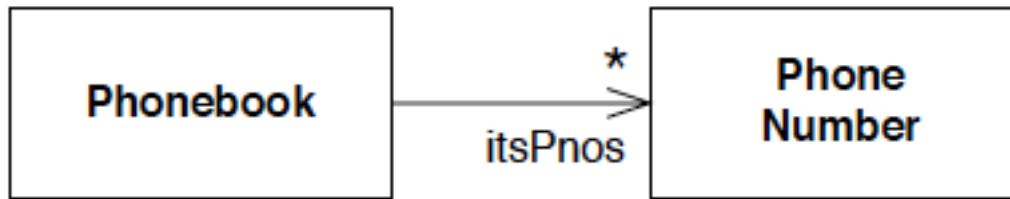
Associazione - la direzione della freccia ci dice che il telefono contiene un riferimento a pulsante

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

Rappresentazione in Java della classe Phone



Molteplicità



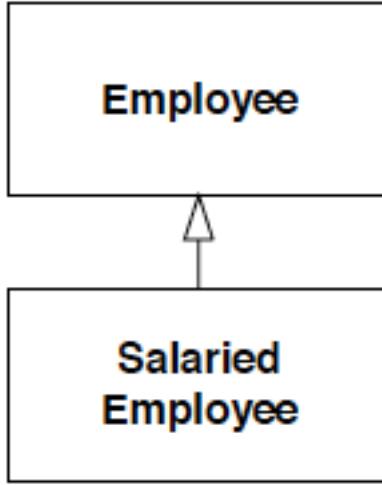
Molteplicità “many”

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

Rappresentazione in Java della classe Phonebook



Ereditarietà



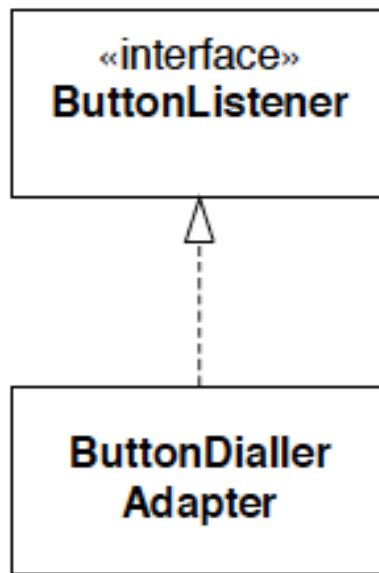
Esempio di ereditarietà

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

Rappresentazione in Java



Interfacce



Implementazione di interfacce

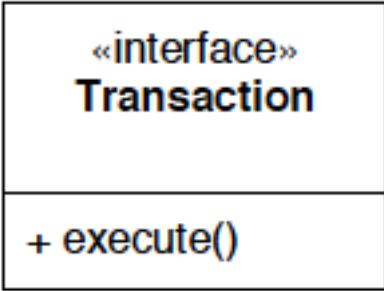
```
interface ButtonListener
{
    ...
}

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

Rappresentazione in Java



Stereotipi di classe



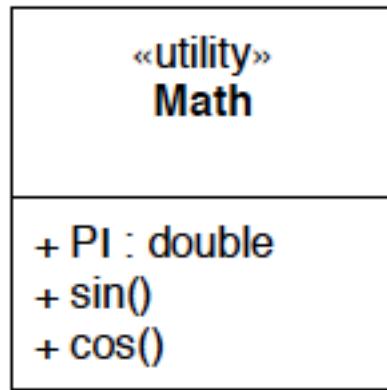
Stereotipo <<interface>>

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

Rappresentazione in Java



Stereotipi di classe



Stereotipo <<utility>>

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

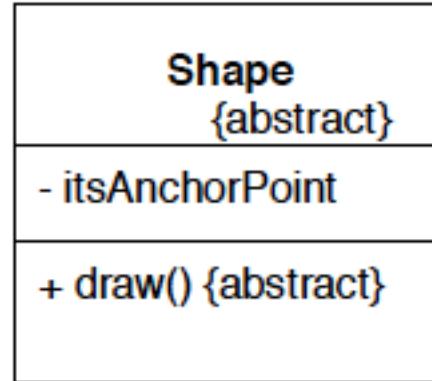
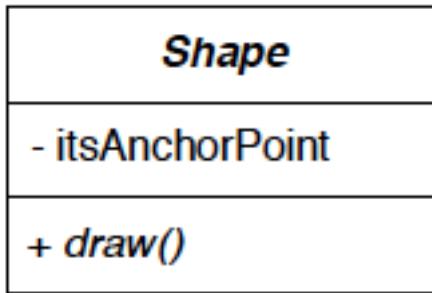
    public static double sin(double theta) { . . . };

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

Rappresentazione in Java



Classi astratte



Rappresentazioni di classi astratte

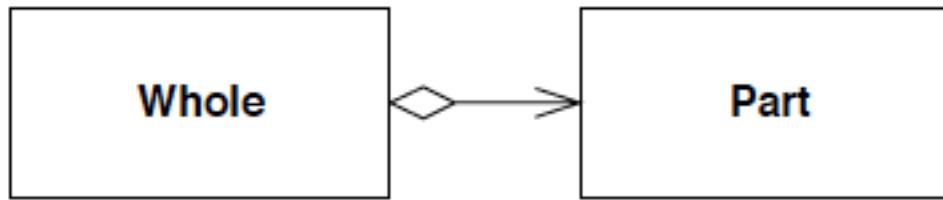
```
public abstract class Shape
{
    private Point itsAnchorPoint;

    public abstract void draw();
}
```

Rappresentazione in Java



Aggregazione



Rappresentazione di aggregazione (Parte-Intera)

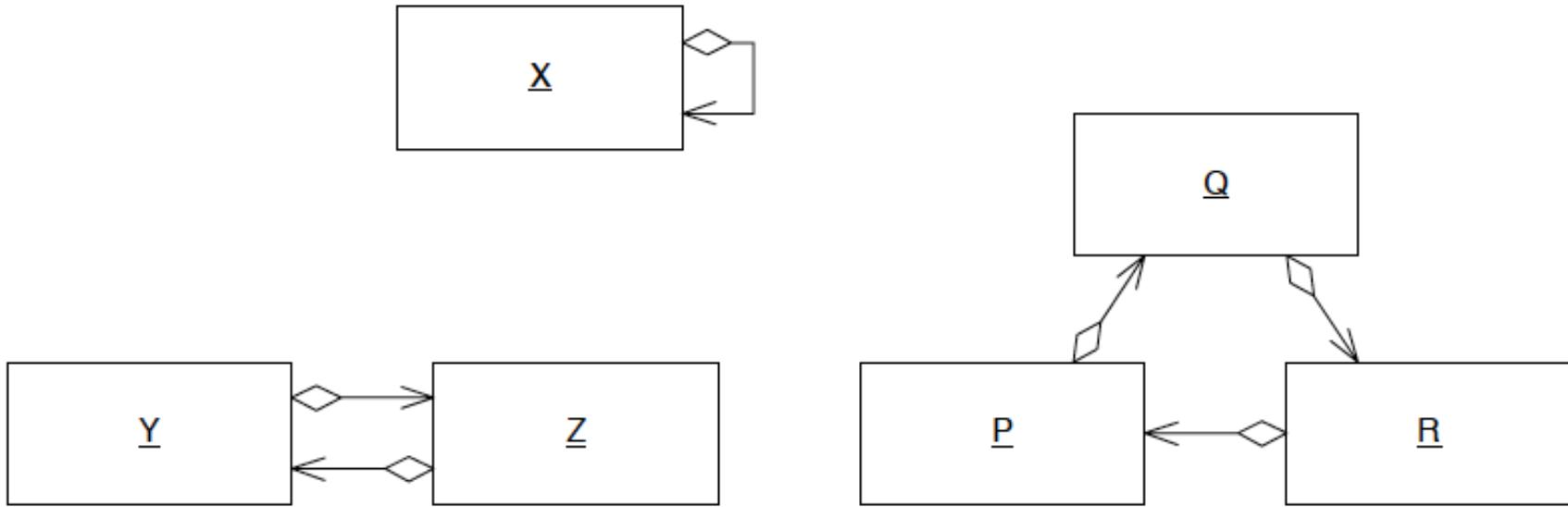
```
public class Whole  
{  
    private Part itsPart;  
}
```

Rappresentazione in Java

Stesso risultato dell'aggregazione



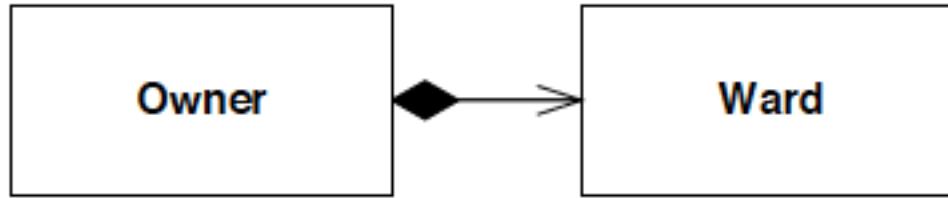
Aggregazione



Aggregazioni non ammesse tra istanze



Composizione



Rappresentazioni di composizione

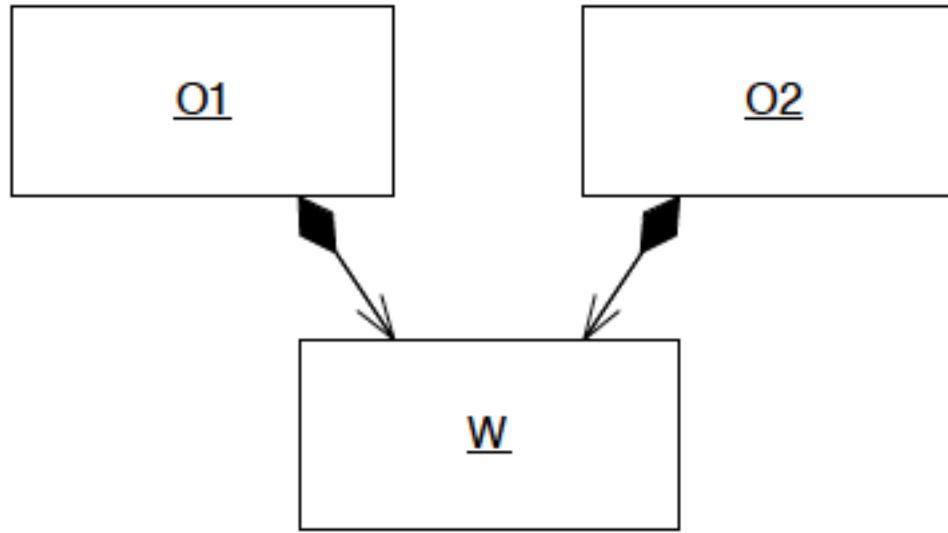
```
public class Owner  
{  
    private Ward itsWard;  
}
```

Rappresentazione in Java

Stesso risultato dell'aggregazione



Composizione

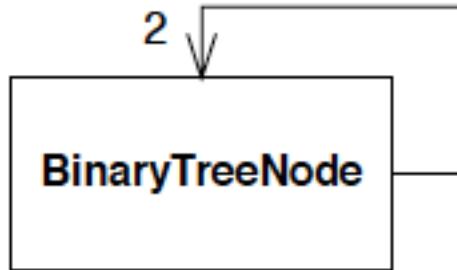


Composizione non ammessa tra istanze – ogni istanza delle classi componenti può appartenere a una sola istanza della classe composta

La classe composta è responsabile della durata della classe composta



Molteplicità

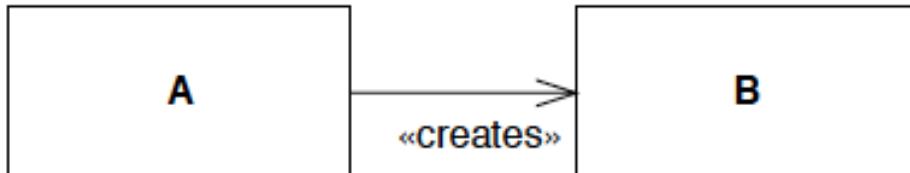


Semplice molteplicità

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

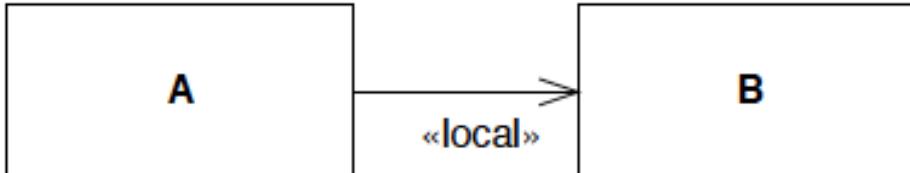


Stereotipi



Stereotipo

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

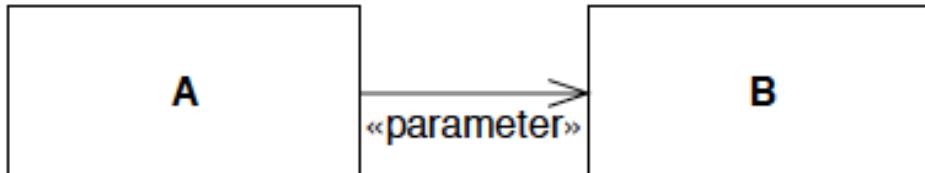


Stereotipo

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



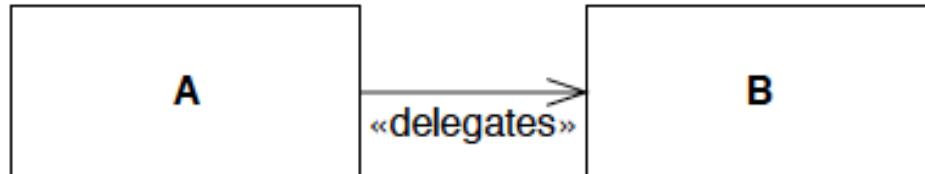
Stereotipi



Stereotipo

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

    }
}
```

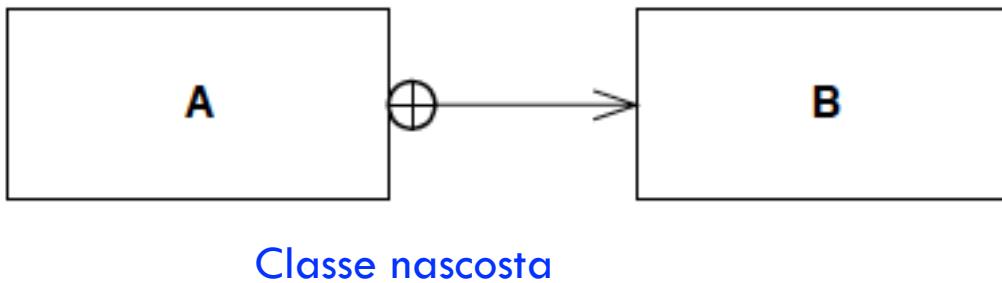


Stereotipo

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

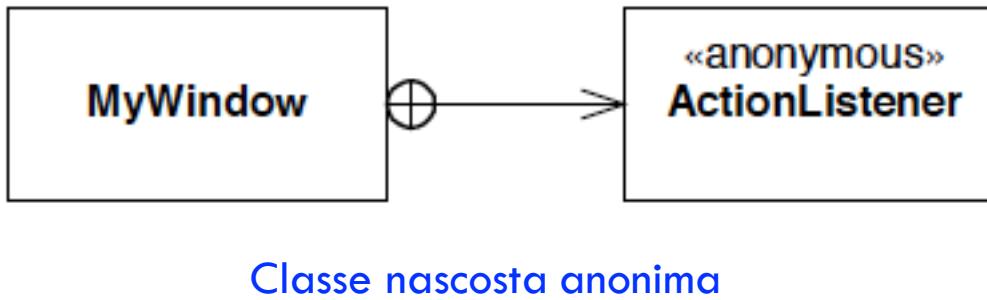


Classi nascoste



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

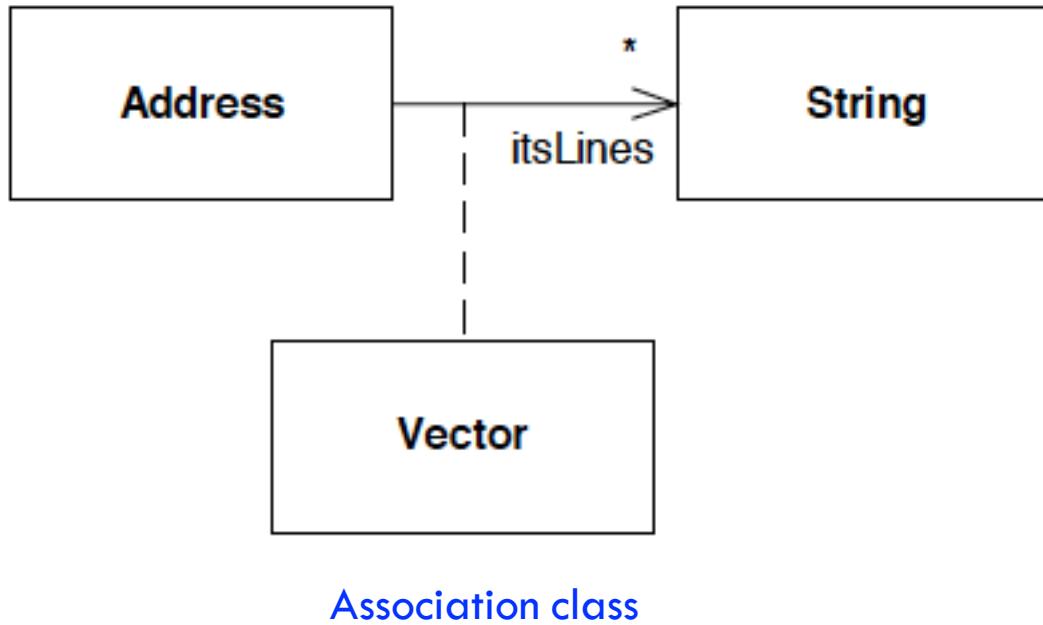
Classi nascoste



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



Association class



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



Software ... Eclipse



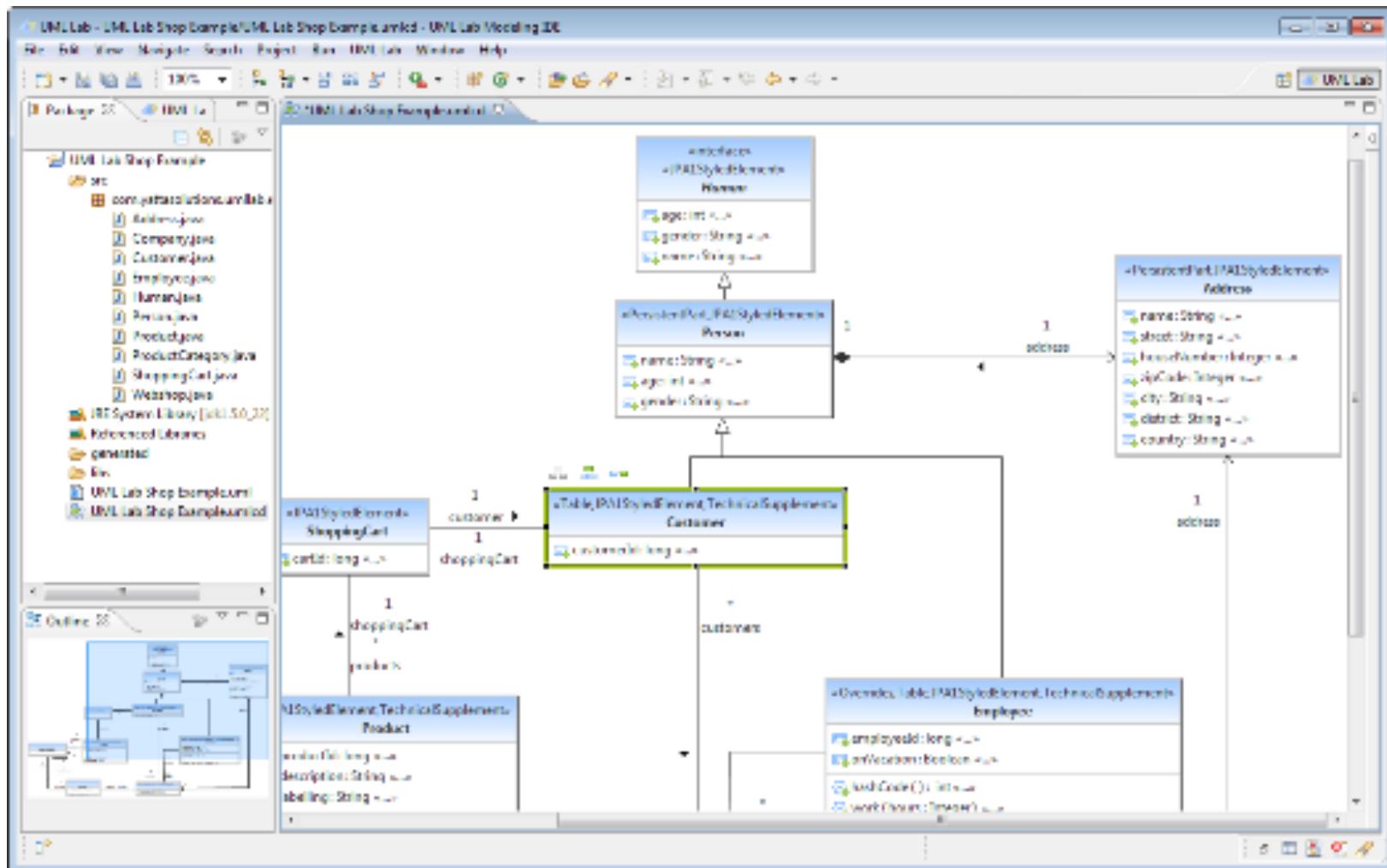
Java - Jedi/src/java/jedi/functional/FunctionalPrimitives.java - Eclipse SDK - /Users/channing/Code/worksheets/workspace3.3

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

Writable Smart Insert 133 : 1

Outline Jedi.jedi.jedi.functional FunctionalPrimitives addToGroup(K, V, Map<K, List<V>>) append(Collection<? extends Collection<T>>, new Functor<Collection<T>>) append(Collection<? extends T>...) collect(Collection<T>, Functor<? super T>) collect(T[], Functor<? super T, R>) dropRight(int, List<T>) dropRight(int, List<T>) flatten(Collection<T>, Functor<? super T>) foldl(Collection<T>, Functor2<R, ?>) forEach(Collection<T>, Command<?>) group(Collection<V>, Functor<? super V>) head(Collection<T>) headOrDefaultEmpty(Collection<T>) headOrDefaultEmpty(Collection<T>, Object) join(Collection<T>, String) listTabulate(int, Functor<Integer, R>) longest(Collection<T>) makeList(int, T) only(Collection<T>) produce(Collection<T>, Collection<U>) random(List<T>) random(T[]) randomOrder(T[]) reverse(List<T>) select(Collection<T>, Filter<? super T>) sequence(Command<? super T>...) new Command<T>() shortest(Collection<T>) slice(int, List<List<T>>) take(int, List<T>) takeRight(int, List<T>) zip(List<List<T>>)

Software ... UML Lab Class Diagrams



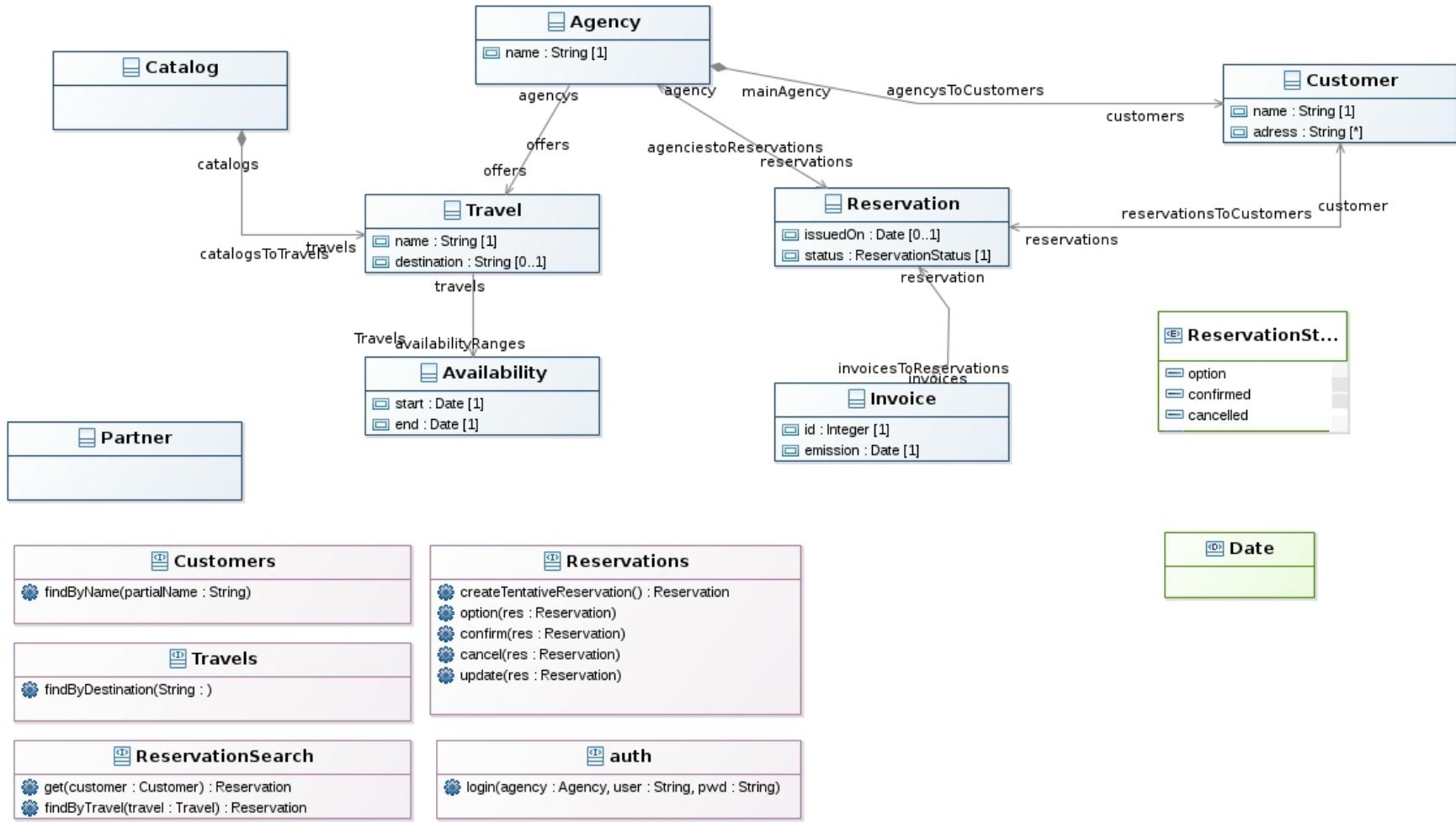
Software ... UML Designer

■ 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

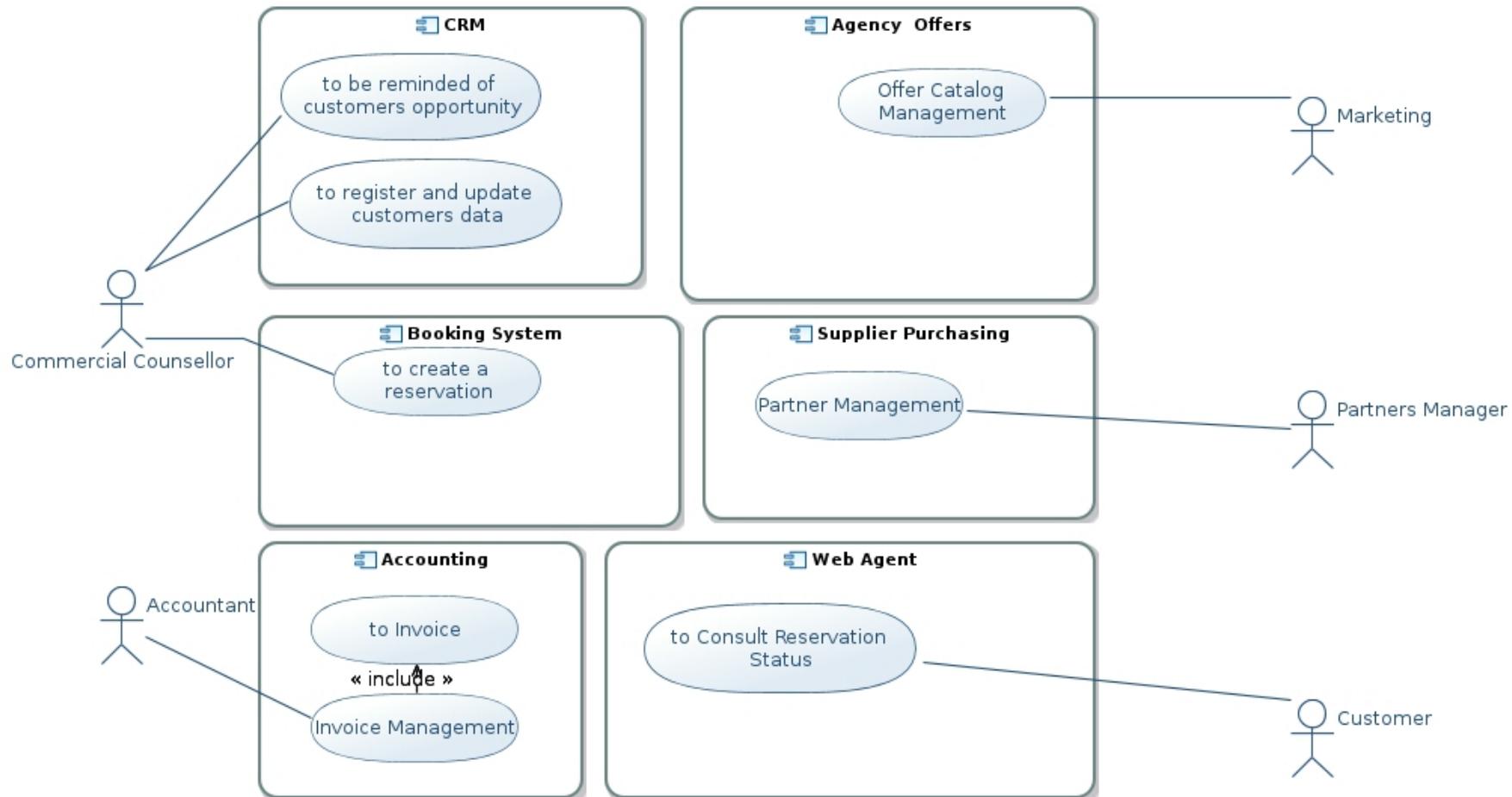


Software ...



Class Diagram

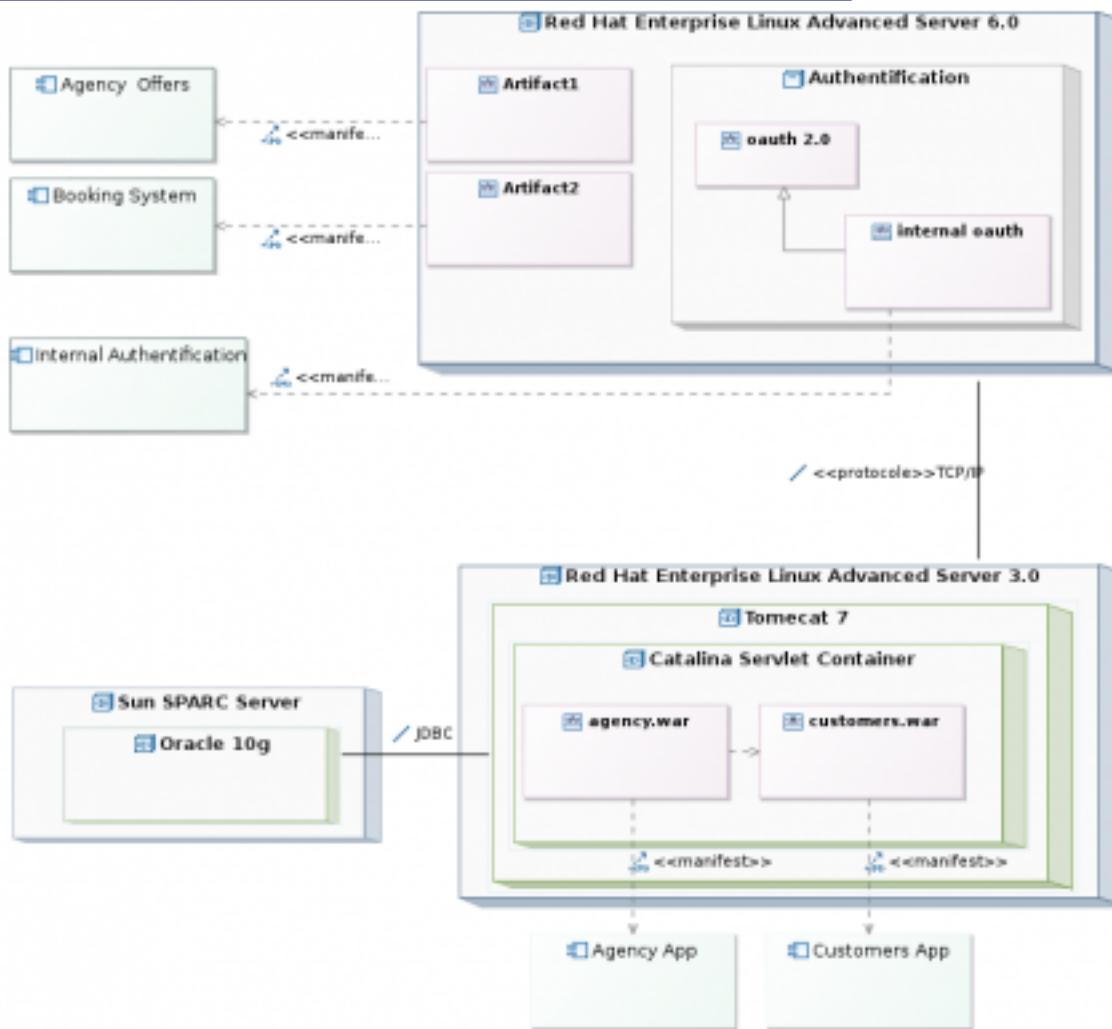
Software ...



Case Use Diagram



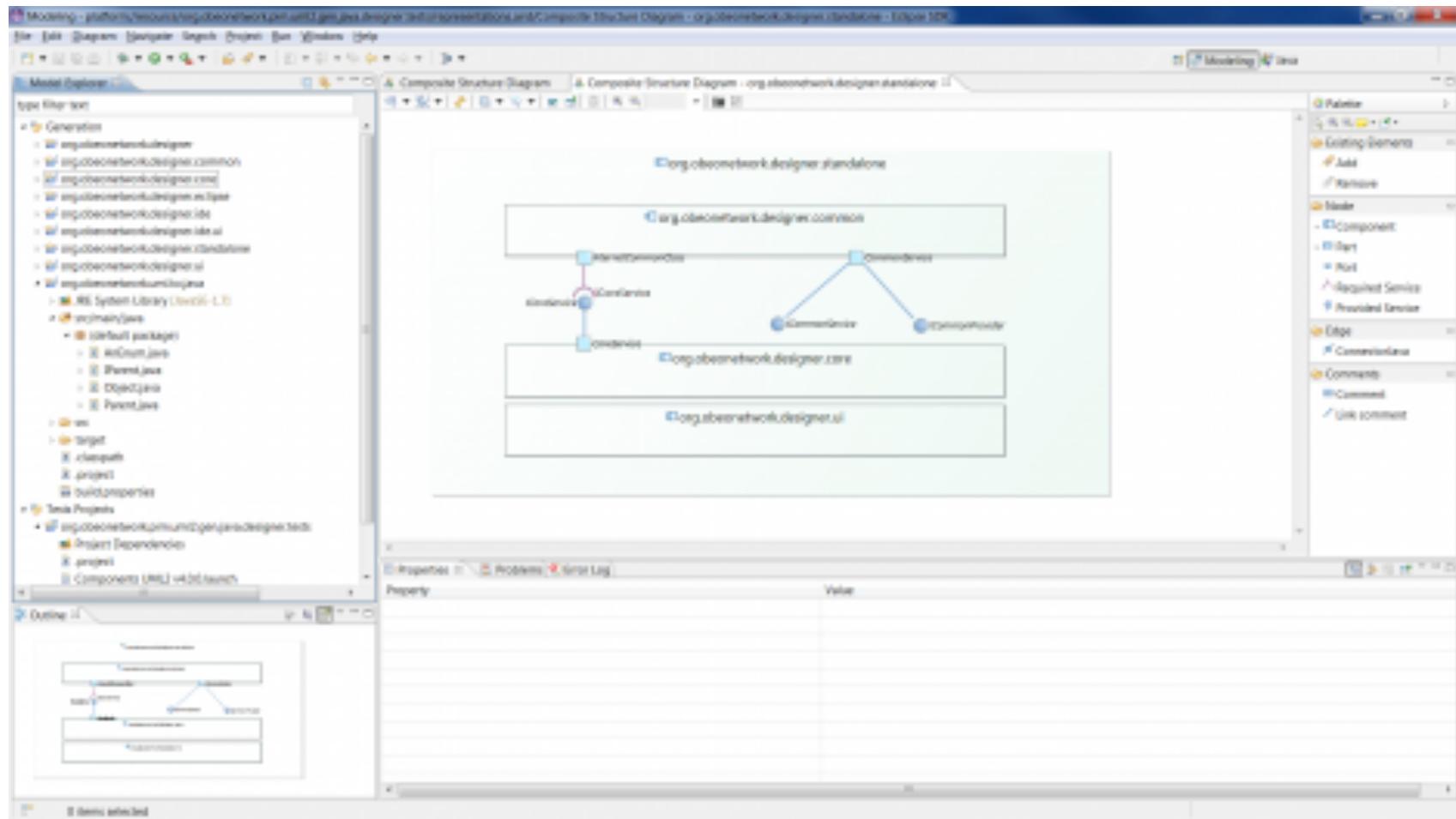
Software ...



Deployment Diagram



Software ... UML to Java Generator



Software ... UML to Java Generator

