The Composite Pattern

This design pattern is called the **Composite Pattern**.

- The recursive object structural design gives rise to recursive algorithms.
- An example is an `ArrayList`.

`composition` in particular, is a common object design.

Often, we combine (or compose) objects to form new objects. Recursive
The Composite Pattern (cont.)
The method \texttt{operation()} for \texttt{Composite} is mostly recursive.

In the previous diagram, classes \texttt{Basic1} and \texttt{Basic2} correspond to the base cases of the recursion, and \texttt{Composite} corresponds to the non-base cases.

\textbf{The Composite Pattern (cont.)}
What is a polynomial?
Polynomials: A UML Diagram
...  
    }
    return -coeff;
  }
  public double getLeadingCoefficient()  
  
  Returns the leading coefficient of this Polynomial.  
  // protected double -coeff;
  
  The leading coefficient of this Polynomial  
  // }
  public abstract class Polynomial

Polynomials class Polynomial
<table>
<thead>
<tr>
<th>Specifier</th>
<th>Class</th>
<th>subclass</th>
<th>package</th>
<th>program</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

**Controlling Access to Members of a Class**
... public abstract double evaluation(double x);

public abstract Polynomial add(Polynomial p);

// Returns the value of this Polynomial at a point x.
// The parameter Polynomial p.
// Returns an Polynomial that is the sum of this Polynomial with

Public abstract class Polynomial

February 1, 2001

Comp 212
... {
    return -coefficient;
}

public double evaluate(double x) {
    // Returns the coefficient of this ConstPoly.
    ...

    {
        coefficient = coefficient;
    }

    public ConstPoly multiply(double coefficient)
    }

    class ConstPoly extends AbstractPoly

    Polynomials class ConstPoly
```java
public NonConstantPoly(double coeff, int degree, Apolyynomial Polynomial) {
    if (coeff == 0) {
        throw new IntegralNonZeroException("coeff must be non-zero!");
    }

    if (null == Polynomial) {
        throw new IntegralNonZeroException("Polynomial must be non-null!");
    }

    // Data Invariant: -degree < degree
    // Data Invariant: 0 > -degree
}
}```
...  

```java
{  
  return (-coef * Math.pow(x, -degree)) + -topRoot.eval(x);
}
```

```java
public double eval(double x) 
  // of the lower-order polynomial at x.  // Evaluates the leading term at x then adds the result to the Val

...

```java
{  
  towerPoly = towerPoly;
  degree = degree;
  coef = coef;

  if (new IllegalArgumentException("TowerPoly is not a Tower of IntegerArgumentException")
    {  
      if (degree <= towerPoly.getDegree())  
        {  
          throw new IllegalArgumentException("Degree must be positive");
        }
      (0 <=
```