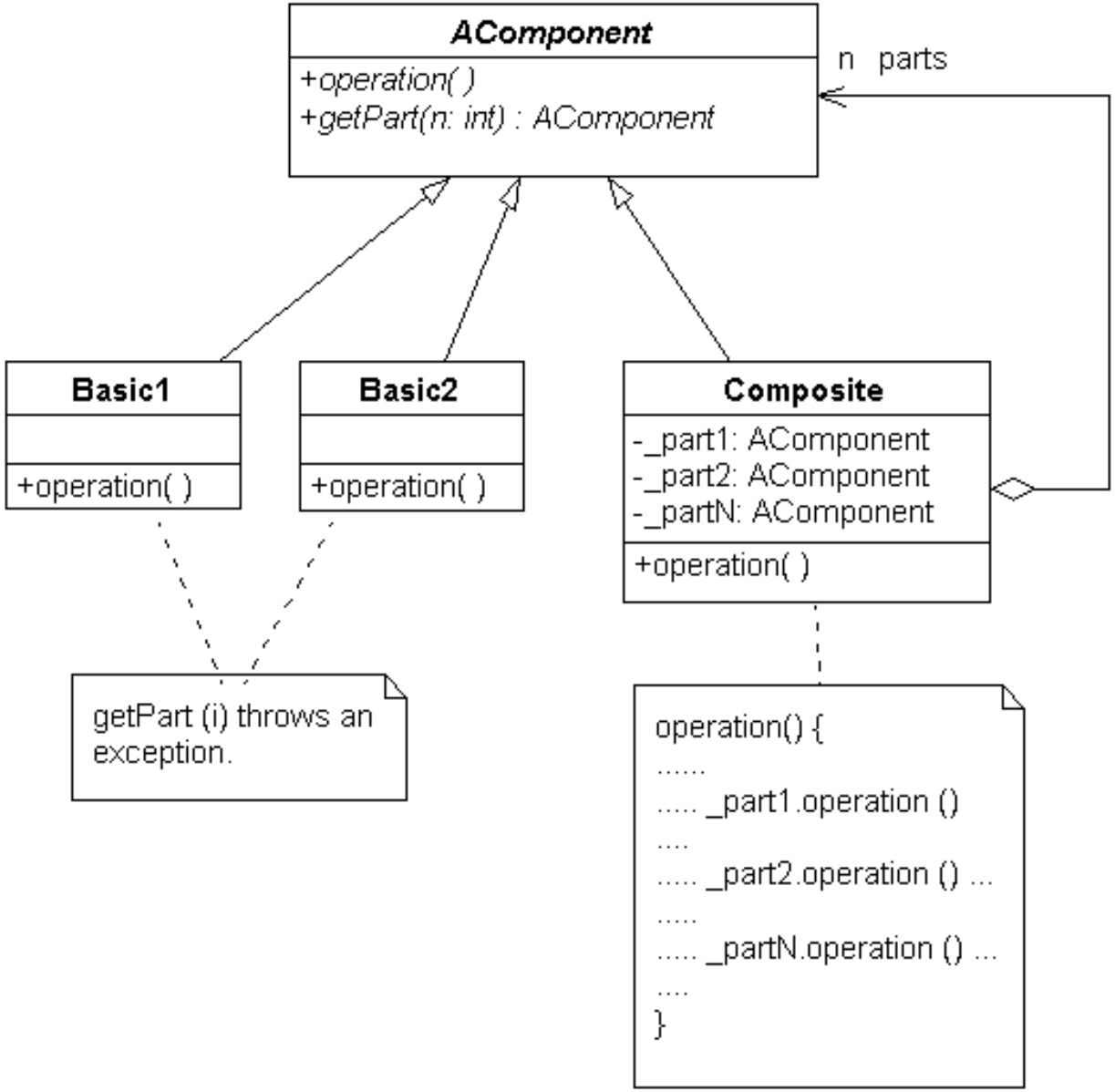
The Composite Pattern

- Often, we combine (or compose) objects to form new objects. Recursive composition, in particular, is a common object design.
  - AList is an example.
  - The recursive object structural design gives rise to recursive algorithms on the object.

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

The Composite Pattern (cont.)



The Composite Pattern (cont.)

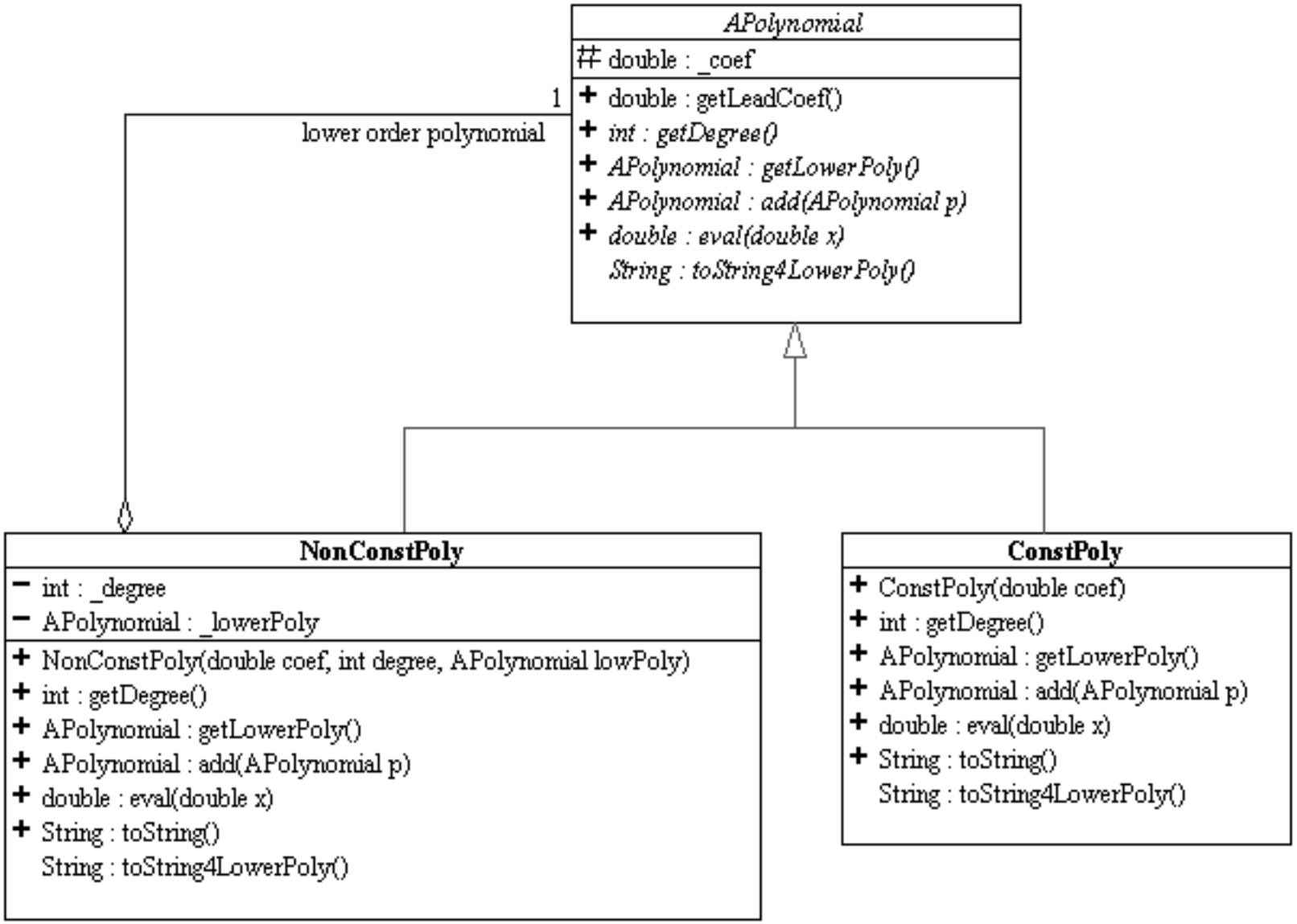
- In the previous diagram, classes `Basic1` and `Basic2` correspond to the base cases of the recursion, and `Composite` corresponds to the non-base cases.

- The method `operation()` for `Composite` is mostly recursive.

Polynomials

- What is a polynomial?

Polynomials: A UML Diagram



Polynomials: `class APolynomial`

```
public abstract class APolynomial
{
  // The leading coefficient of this APolynomial
  protected double _coef;

  // Returns the leading coefficient of this APolynomial.
  public double getLeadCoef()
  {
    return _coef;
  }
  ...
```

Controlling Access to Members of a Class

Specifier	class	subclass	package	program
private	X			
protected	X	X	X	
	X		X	
public	X	X	X	X

Polynomials: `class APolynomial`

```
public abstract class APolynomial
{
  ...
  // Returns an APolynomial that is the sum of this APolynomial wit
  // the parameter APolynonial p.
  public abstract APolynomial add(APolynomial p);

  // Returns the value of this APolynomial at a "point" x.
  public abstract double eval(double x);
  ...
```

Polynomials: `class ConstPoly`

```
class ConstPoly extends APolynomial
{
  public ConstPoly(double coef)
  {
    _coef = coef;
  }
  ...
  // Returns the coefficient of this ConstPoly.
  public double eval(double x)
  {
    return _coef;
  }
  ...
```

Polynomials: class NonConstPoly

```
class NonConstPoly extends APolynomial
{
  // Data Invariant: 0 < _degree
  private int _degree;

  // Data Invariant: _lowerPoly._degree < _degree
  private APolynomial _lowerPoly;

  public NonConstPoly(double coef, int degree, APolynomial lowPoly)
  {
    if (null == lowPoly) {
      throw new IllegalArgumentException("lowPoly must be non-null
    }
    if (0.0 == coef) {
      throw new IllegalArgumentException("coef must be non-zero!");
    }
```

```
    if (degree <= 0) {
      throw new IllegalArgumentException("degree must be positive!"
    }
    if (degree <= lowPoly.getDegree()) {
      throw new IllegalArgumentException("lowPoly is not a lower or
    }
    _coef = coef;
    _degree = degree;
    _lowerPoly = lowPoly;
  }
  ...
  // Evaluates the leading term at x then adds the result to the va
  // of the lower-order polynomial at x.
  public double eval(double x)
  {
    return (_coef * Math.pow(x, _degree)) + _lowerPoly.eval(x);
  }
  ...
```