Design Patterns

- The Union Pattern
- The Strategy Pattern
- The Composite Pattern

Object-oriented Programming Principles

1. Objects are the only things that can perform computations.

2. Encapsulate that which varies (a variant) into a class, and make all related variants into concrete subclasses of an "abstract class".

   - E.g., `Rectangle` and `Circle` extend `Shape`.

3. Program to the interface (or abstract class).

   - E.g., `_shape.dArea();` where `_shape` is `AShape`, not `Rectangle` or `Circle`.

The Union Pattern

- Suppose I face the problem of computing the areas of geometrical shapes such as rectangles and circles.
- OOPP #0 suggests that I build objects that are capable of computing these areas.
- The variants for this problem are the infinitely many shapes: rectangles, circles, etc.
  - OOPP #1 drives me to define concrete classes such as Rectangle and Circle, and make them subclasses of an abstract class, called AShape, which has the abstract capability of computing its area.
    - This is an example of the simplest yet most fundamental OO design pattern called the *Union Pattern*. It is the result of applying OOPP #0 and OOPP #1.

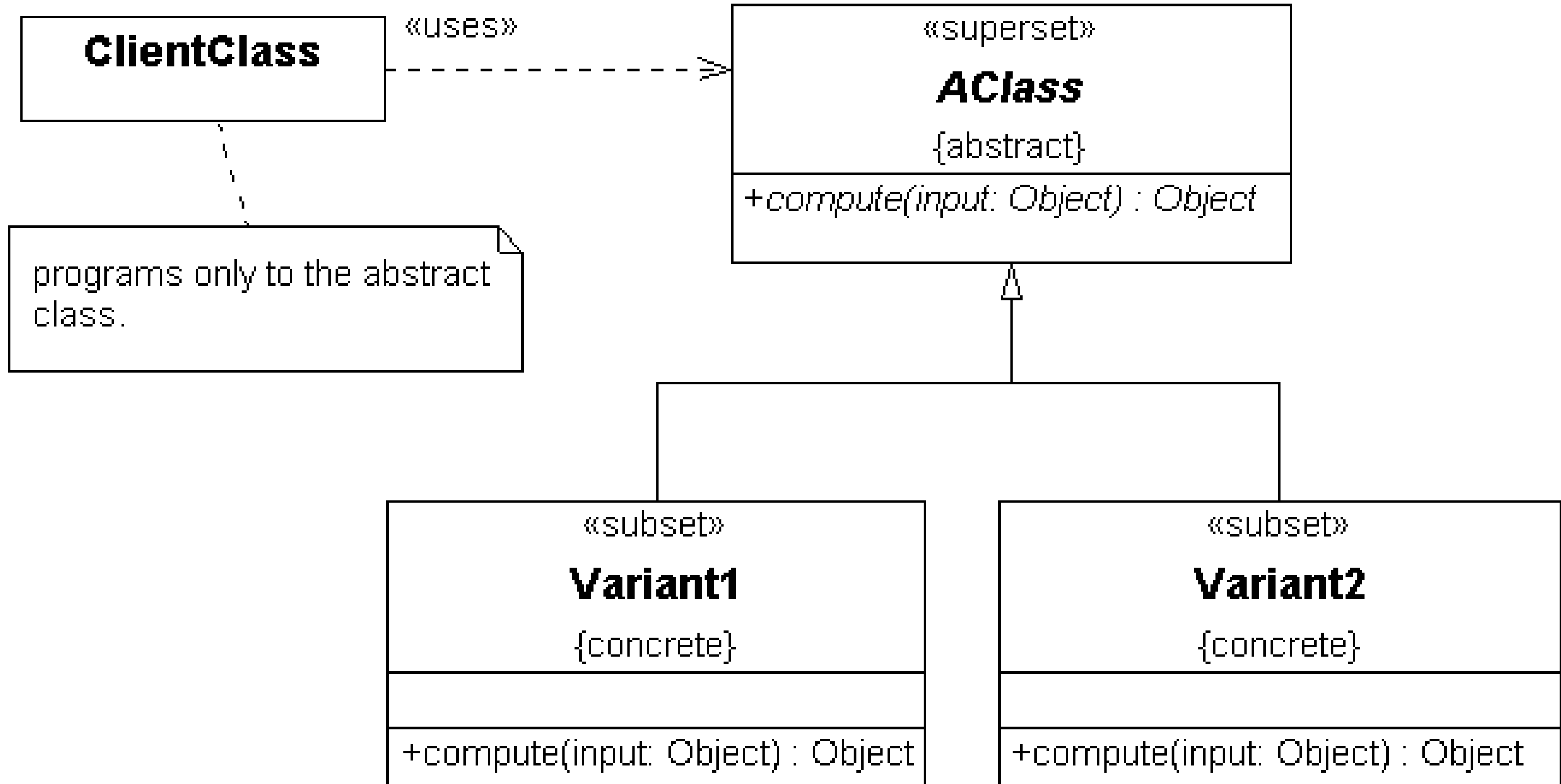
The Union Pattern (cont.)

- The Union Pattern is the result of *partitioning* the sets of objects in the problem domain into *disjoint* subsets and consists of
  - an abstract class (AClass) representing the superset of all the objects of interest,
  - several concrete subclasses (Variant1, Variant2) representing disjoint subsets of the above superset;
    - the union of these subsets equals the superset.

The Union Pattern (cont.)

- A client of the Union Pattern uses instances of the concrete subclasses (Variant1, Variant2), but should only see them as AClass objects.
  - The client class code should only concern itself with the public methods of AClass and should not need to check for the class type of the concrete instances it is working with.
    - Conditional statements to distinguish the various cases are gone, reducing code complexity and making the code easier to maintain.

The Union Pattern (cont.)



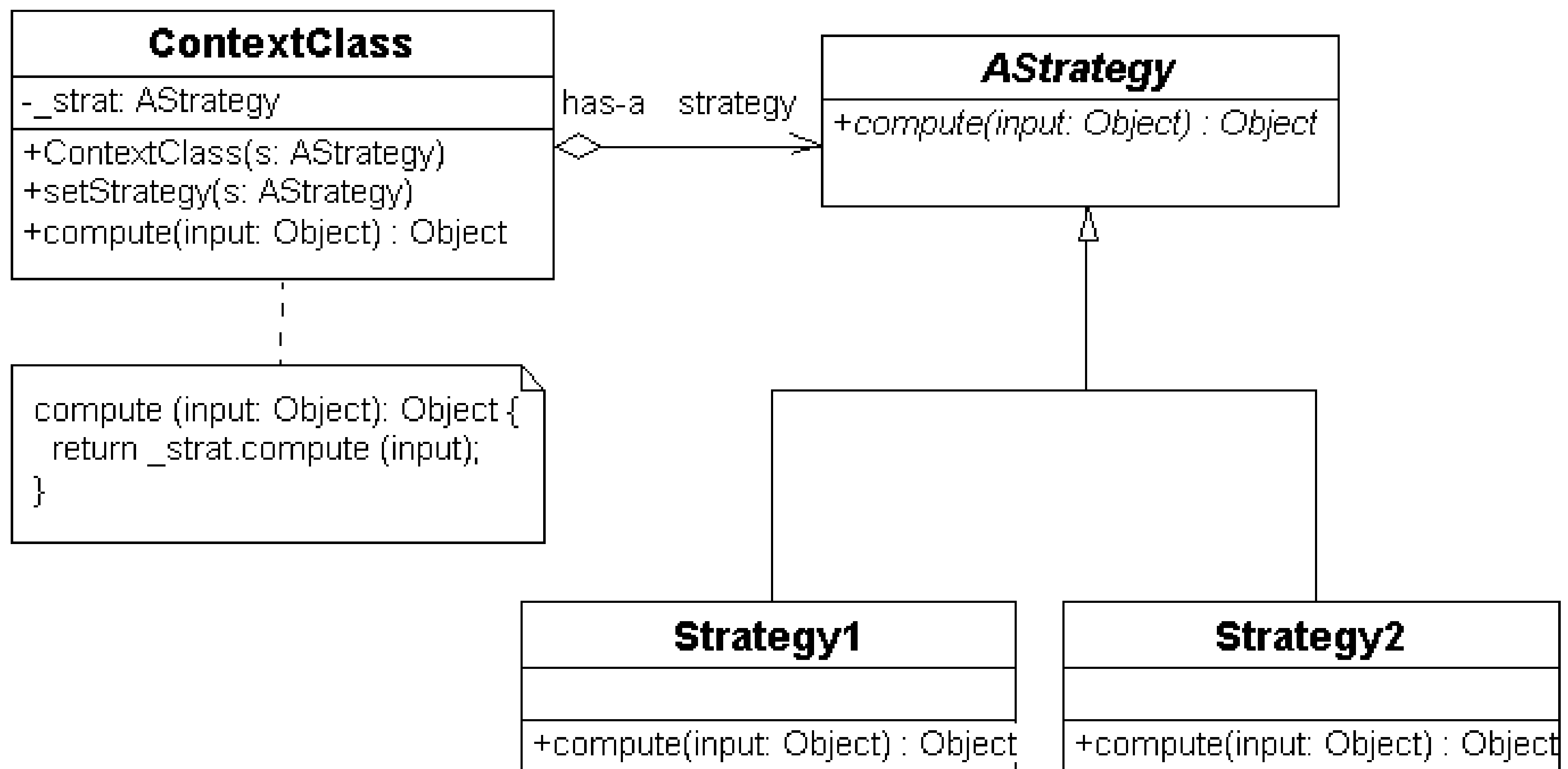
The Strategy Pattern

- Recall the Pizza problem. The Pizza has a shape and delegates the computation of its area to its shape.
  - It does not care what the exact type of its shape is.
  - It only knows that its shape is capable of computing the appropriate area.
  - This is an example of what is called the Strategy Pattern.
    - The Pizza uses its shape as a "strategy" to compute its area.

The Strategy Pattern (cont.)

- In general, the strategy pattern consists of a union pattern of strategies, and a client class, the *context*, that contains a reference to the abstract strategy in the union.
- The context delegates the work to this stategy reference.
  - In our Pizza example, the context is the Pizza class, and the abstract AShape plays the role of the (abstract) strategy.

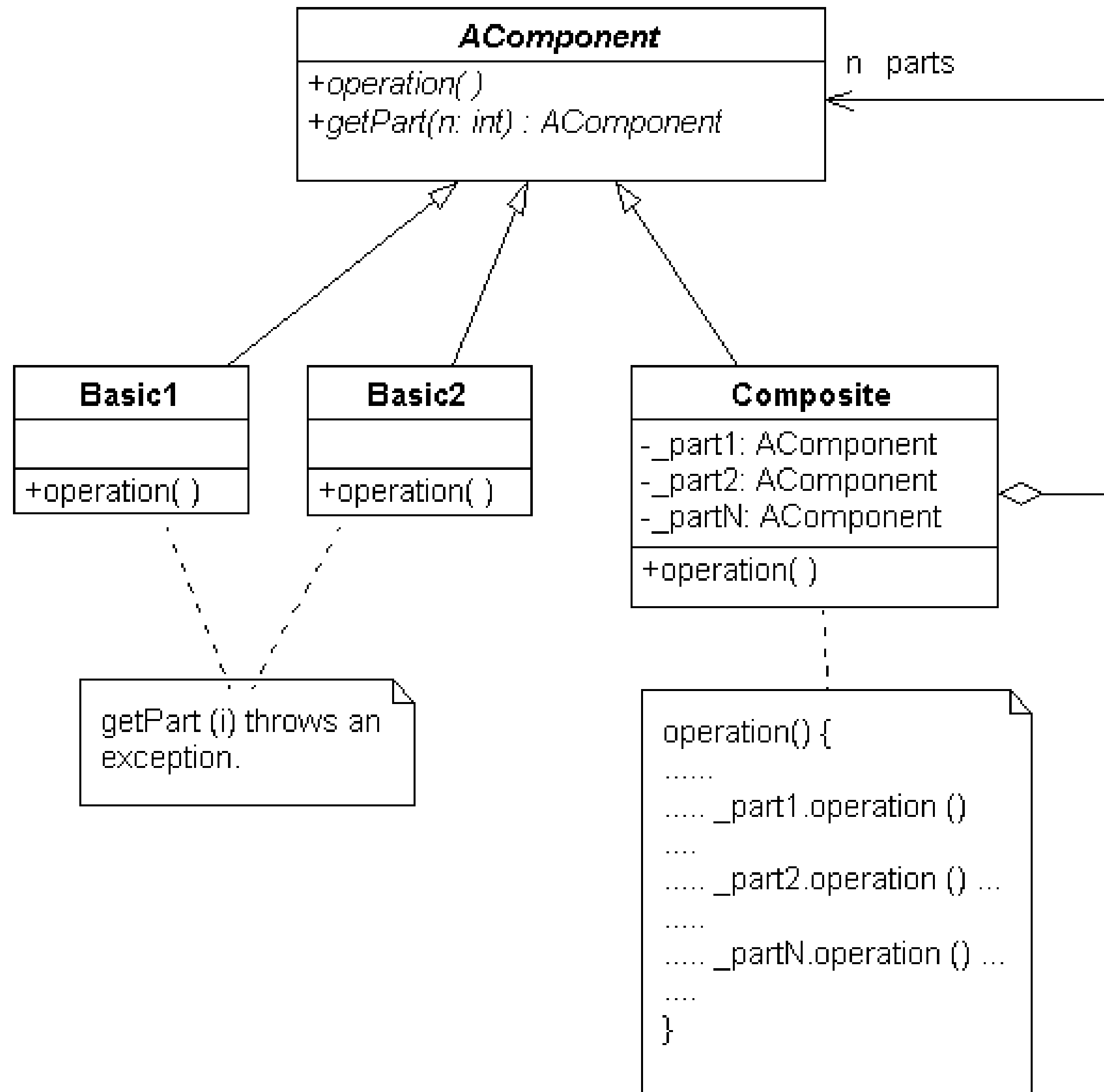
The Strategy Pattern (cont.)



The Composite Pattern

- Often, we combine (or compose) objects to form new objects. Recursive composition, in particular, is a common object design.
  - APolynomial 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.