

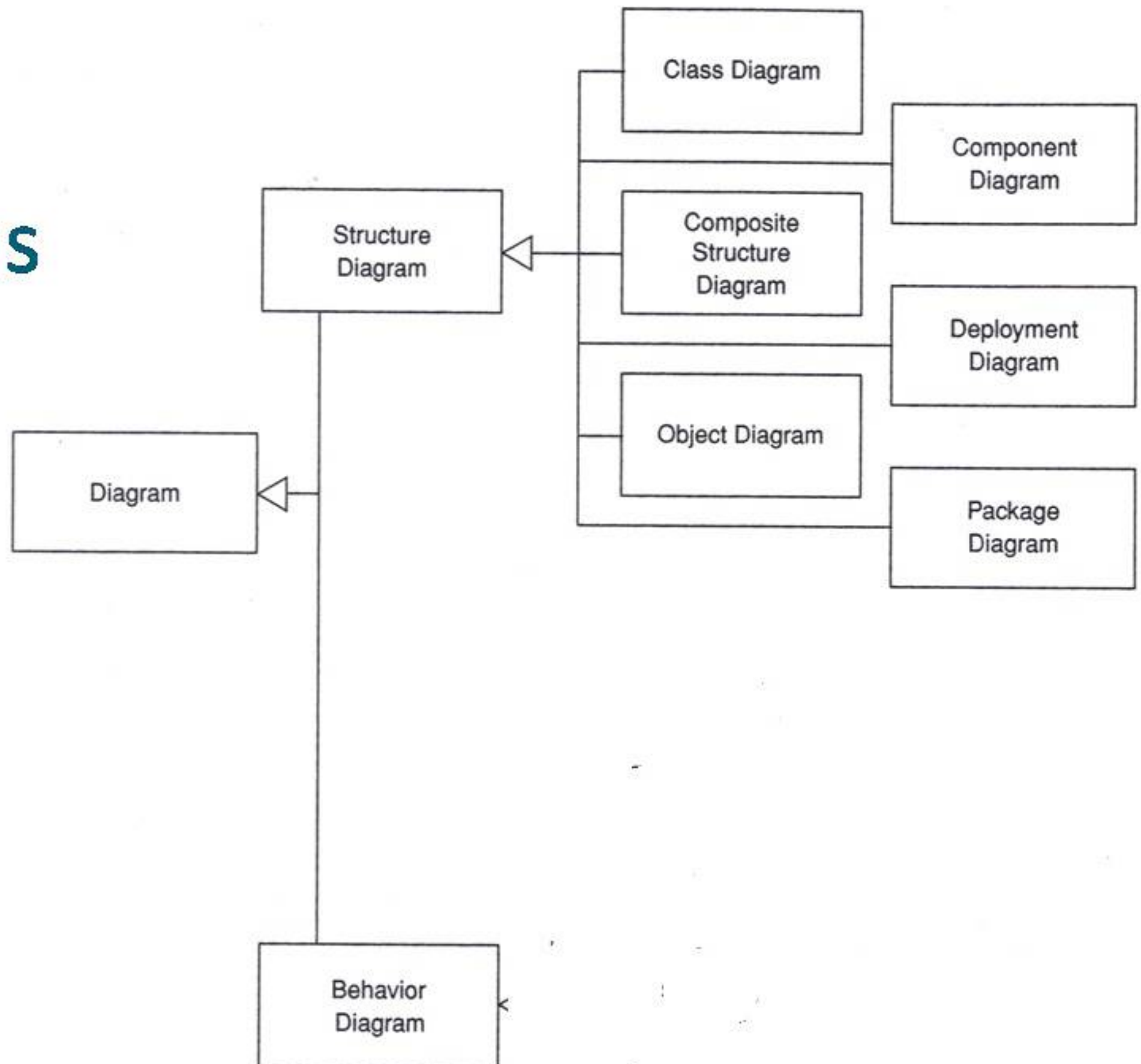


Introduction to UML

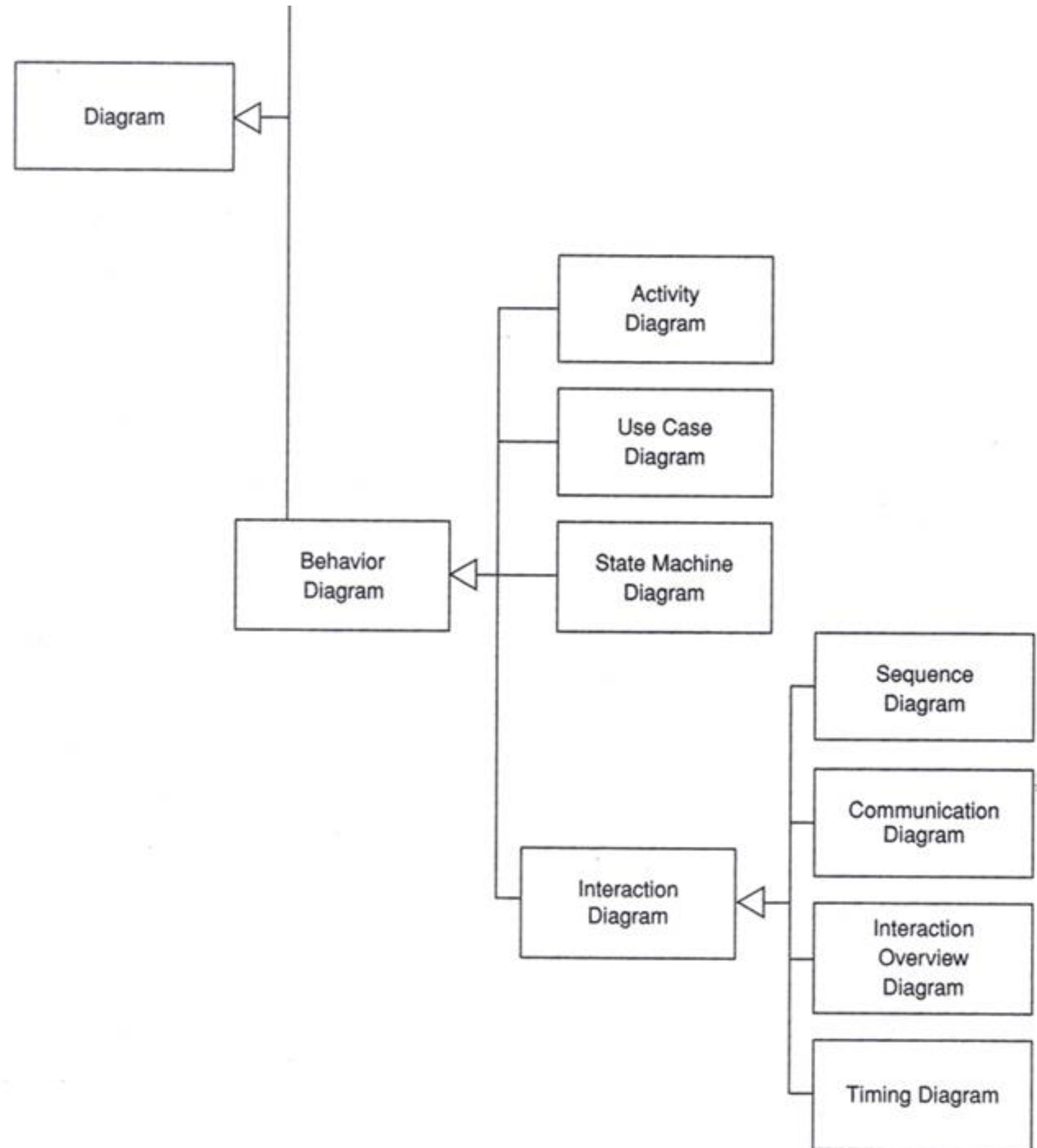
UML Diagrams

- UML 2.0 supports 13 different types of diagrams
- Each diagram may be expressed with varying degrees of detail
- Not all diagrams need be used to model a SW system
- The UML does not offer an opinion as to which diagrams would be most helpful for a particular type of project

UML Diagrams



UML Diagrams



What is Legal UML?

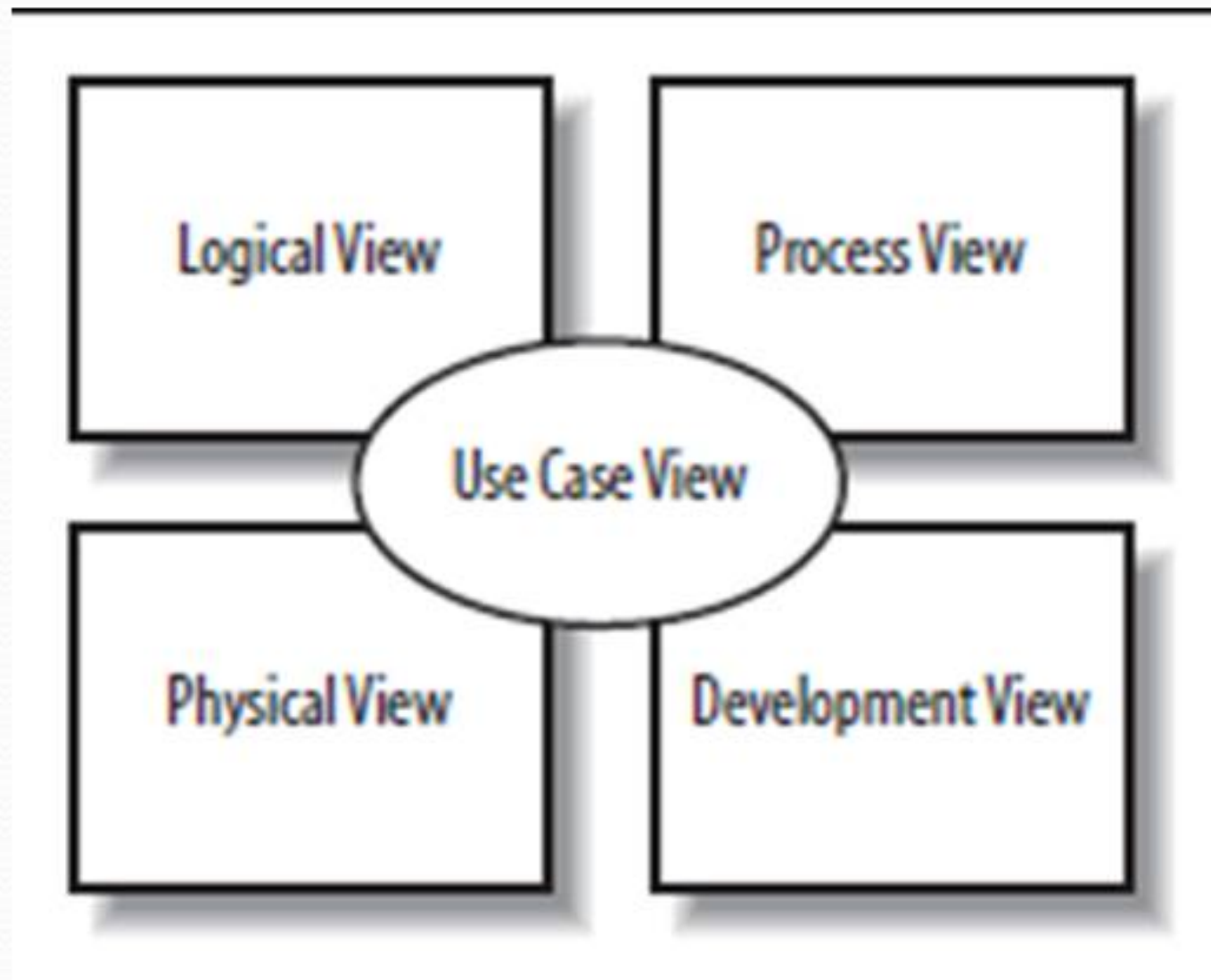
- Even though UML is standardized by the OMG developers take liberties with syntax.
- These “liberties” have a way of becoming standard conventions in later releases of the language.
- The same holds true for natural languages.
- Even if the UML is your primary modeling language, don't hesitate to use other diagrams to model your design.

Architecture

- A complex system needs to be viewed from different perspectives
- Architecture is the set of decisions about
 - The organization of a software system
 - The selection of the structural elements and their interfaces
 - Their behavior
 - The composition of these structural and behavioral elements into progressively larger subsystems

Architecture

- Best described by five (4+1) interlocking views:



Philippe Kruchten's 4+1 view model



Logical view


Describes the abstract descriptions of a system's parts (class, object, state machine, and interaction diagrams).

Process view

Describes the processes within your system (activity diagrams).

Development view

Describes how system's parts are organized into modules and components (package and component diagrams.)



Physical view

Show how the abstract parts map into the final deployed system (deployment diagrams).

Use case view

Describes the functionality of the system being modeled from the perspective of the outside world. All of the other views rely on the use case view to guide them (use case diagrams, descriptions, and overview diagram)

Software Development Life Cycle

- UML is involved in each phase of the software development life cycle.
- The UML development process is
 - Use case driven
 - Architecture-centric
 - Iterative and incremental

Class Diagram

- **Describes the types of objects in the system and the various kinds of static relationships that exist among them**
- **Shows the features of a class and the constraints that apply to the way objects are connected.**
- **Feature:** properties or operations of a class

Terms and Concepts

- *A class is ...*
 - The most important building block of any object-oriented system.
 - A blueprint for creating an object.
 - An abstraction (simplification) of some reality.
 - A representation of a software thing, a hardware thing, or even a conceptual thing.
 - **Graphically represented as a rectangle.**

Terms and Concepts

- *All classes have ...*
 - Names - Used to distinguish one class from another. **A class must have a name.**
 - Attributes - Member data
 - Operations (behaviors) - Member functions (C++) or methods (Java)

Window
origin size
open() close() move() display()

Terms and Concepts

- *Class names are ...*
 - Textual strings (any number of letters, numbers, and certain punctuation marks)
 - Simple or Qualified
 - extracted from the problem domain (statement)
 - nouns or noun phrases
 - concise and descriptive
- Capitalize the first letter of every word in a class name (TemperatureSensor, Customer)

Terms and Concepts

- *Class attributes are ...*
 - usually nouns or noun phrases.
 - Extracted from the problem statement.
 - used to represent properties of the enclosing class
 - determine the state of an object
- Capitalize the first letter of every word in an attribute name except the first letter

Terms and Concepts

- usually notated using the format
 - [visibility] name [: type] [multiplicity]
[= default] [{property -string}]
- Example:

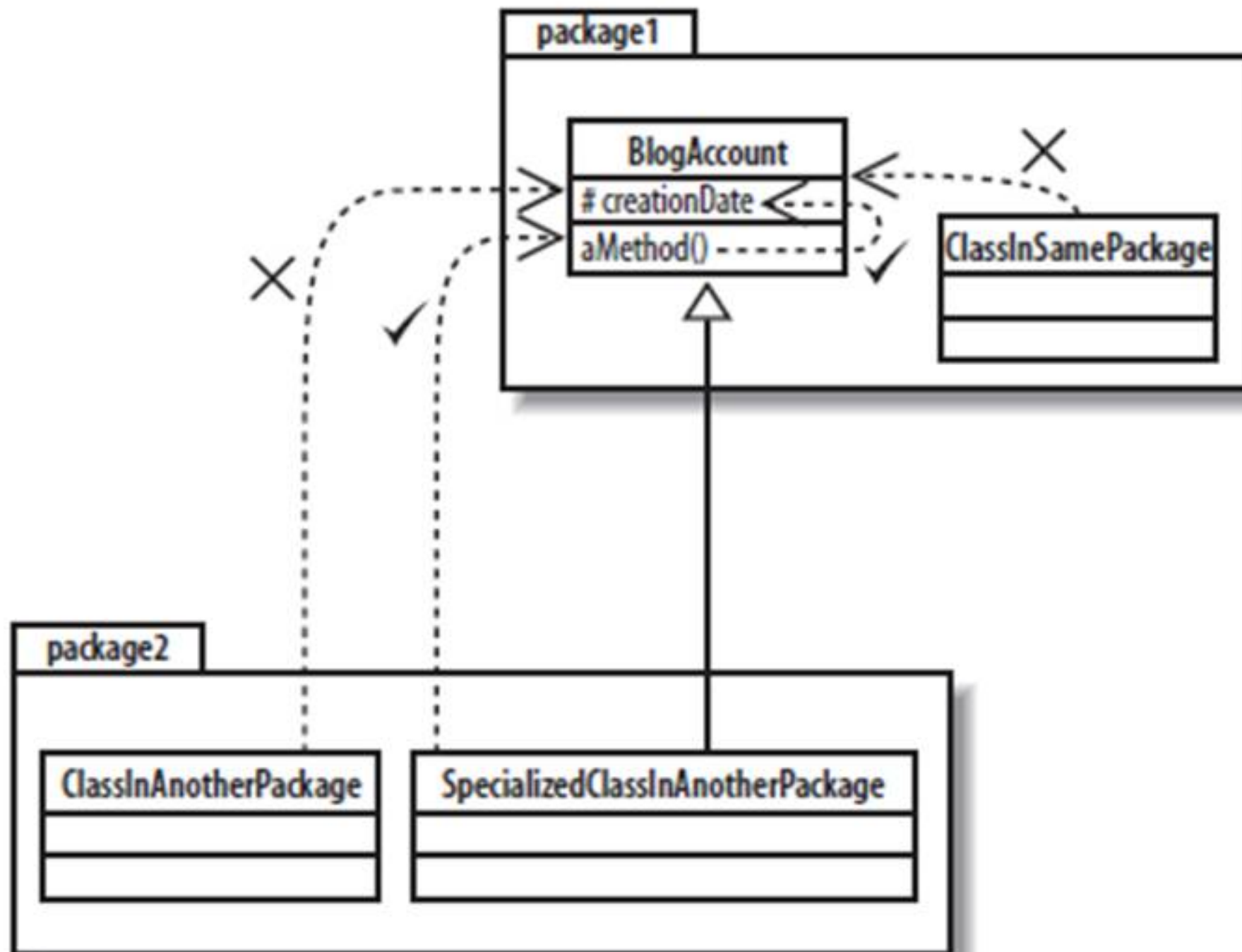
 firstName : String = “Bob”{read only}

 authors : Author [1..5]
- optionally notated as associations

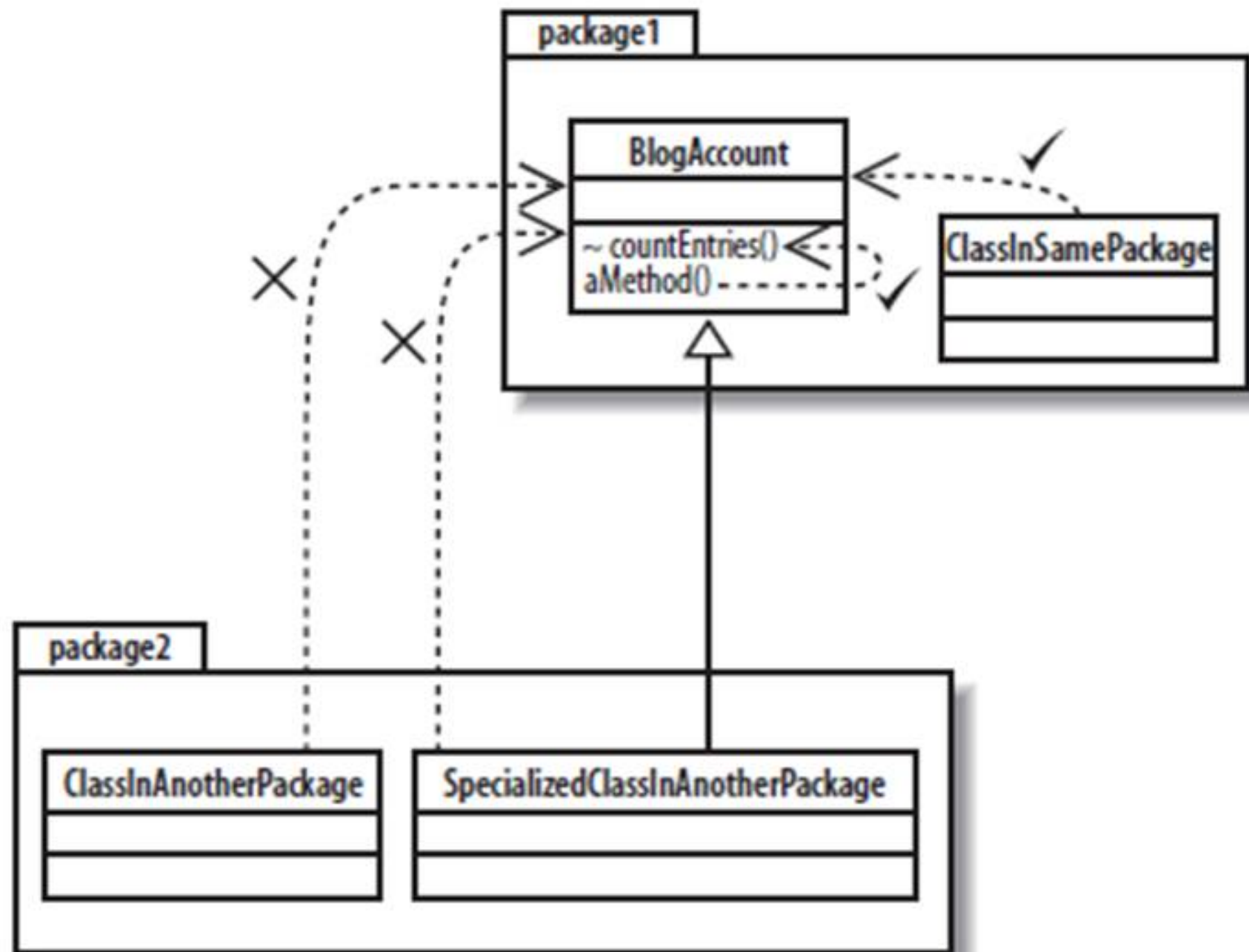
Visibility


- public (+), private (-), protected (#) or package (~)
- Single character visibility indicator
 - # toolCount : Integer
 - - name : String [0..1]
 - + origin
- Restricting visibility is the same as restricting accessibility
- Visibility is used to hide implementation details and expose only those features that are necessary to carry out the responsibilities of the abstraction

Protected



Package






C++

- **private:** Its name can be used only by member functions and friends of the class in which it is declared
- **protected:** Its name can be used only by the member functions and friends of the class in which it is declared and by member functions and friends of classes derived from this class
- **public:** name can be used by any function


```
class Vector{  
    float v[4];  
public:  
    Vector(){  
        v[0] = 1; v[1] = 2; v[2] = 3; v[3] = 4; }  
    void display(){  
        for(int i = 0; i < 4; i++) cout << v[i] << endl;  
    }  
    friend Vector mul(const Matrix&, const Vector&);  
};  
class Matrix{  
    Vector v[4];  
public:  
    friend Vector mul(const Matrix&, const Vector&);  
};
```

```
Vector mul(const Matrix& m, const Vector& v) {  
    Vector r;  
    for (int i = 0; i < 4; i++) {  
        r.v[i] = 0;  
        for (int j = 0; j < 4; j++) r.v[i] += m.v[i].v[j] *v.v[j];  
    }  
    return r;  
}  
  
int main(){  
    Vector myV, myV2;  
    Matrix myM;  
    myV2 = mul(myM, myV);  
    myV2.display();  
}
```

An ordinary member function declaration specifies three logically distinct things:

[1] The function can access the private part of the class declaration and

[2] the function is in the scope of the class and

[3] the function must be invoked on an object (has a this pointer)

By declaring a member function static, we can give it the first two properties only. By declaring a function a friend, we can give it first property only

Instance & Static Scope

- Individual member data (attributes) may have either *static* scope or *instance* scope.
- Static scope - A single copy of an attribute is shared by all instances of a class.
- Instance scope - Each instance of a class would have its own copy of the attribute.

