

Introduction to UML

Unified Modeling Language

Method: An Object Oriented Incremental and Iterative method is a set of guidelines and suggestions (process) about how to structure one's thinking while building a software system. A method guides the software architect :

through various **phases** of software development (**analysis, design, implementation, testing**),

about which **models** to build, when to build them and why,

about how to develop software systems **incrementally** and **iteratively**.

A method uses a modeling language for model building

Modeling Language: A model is expressed in a Modeling Language. A Modeling Language consists of notation for **diagrams** - the symbols used in the models - and a set of rules directing how to use it.

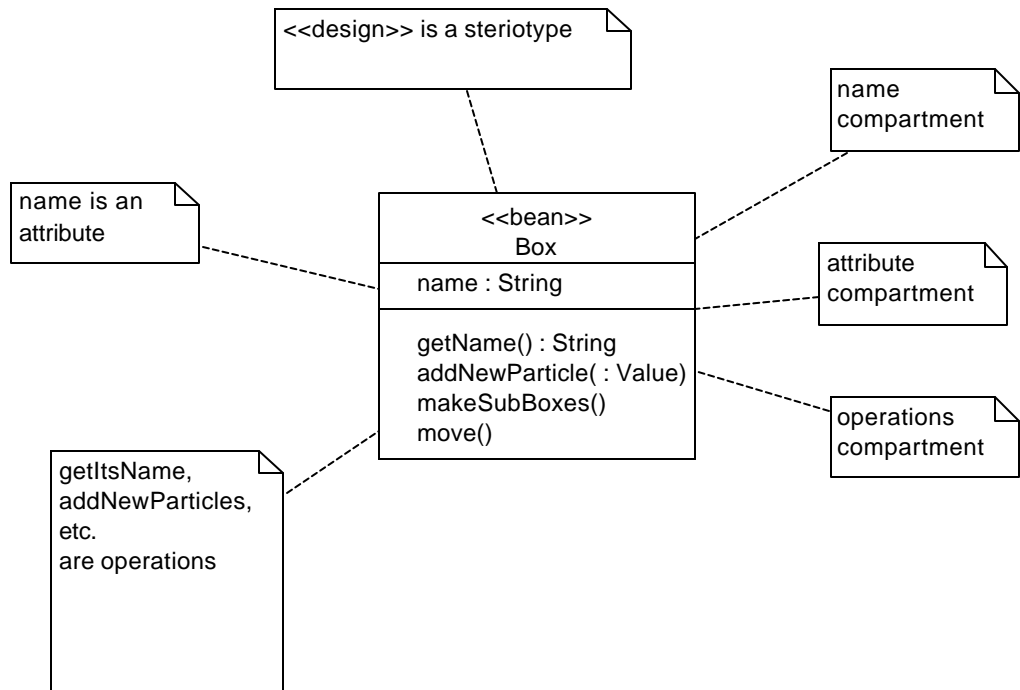
Computer Aided Software Development (CASE) Tools: Software in which we can directly input our (requirements, analysis and design) models in terms of diagrams based on a Modeling Language. A CASE tool:

automatically converts these models into code for the desired language of implementation (e.g. Java or C++).

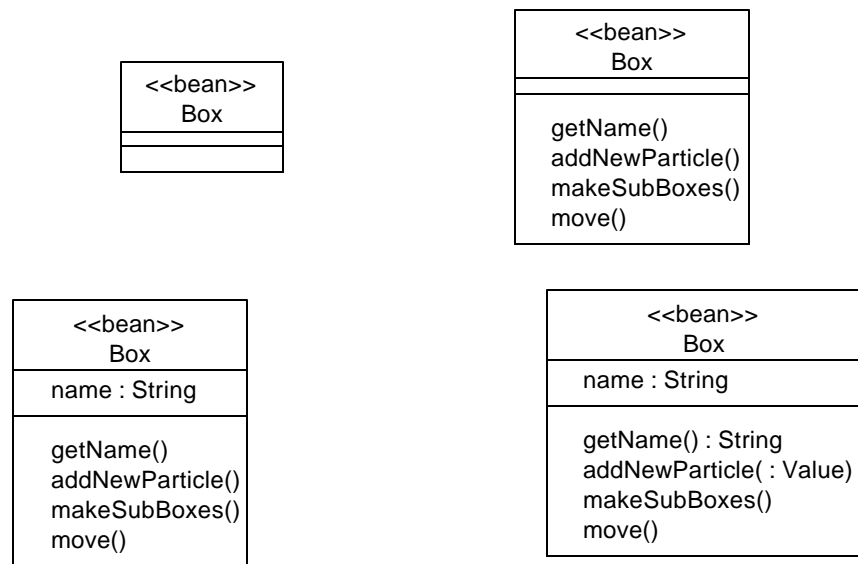
makes sure that all the diagrams are **consistent**.

can convert existing implementation code into diagrams based on a Modeling Language. (**Round Trip Engineering**).

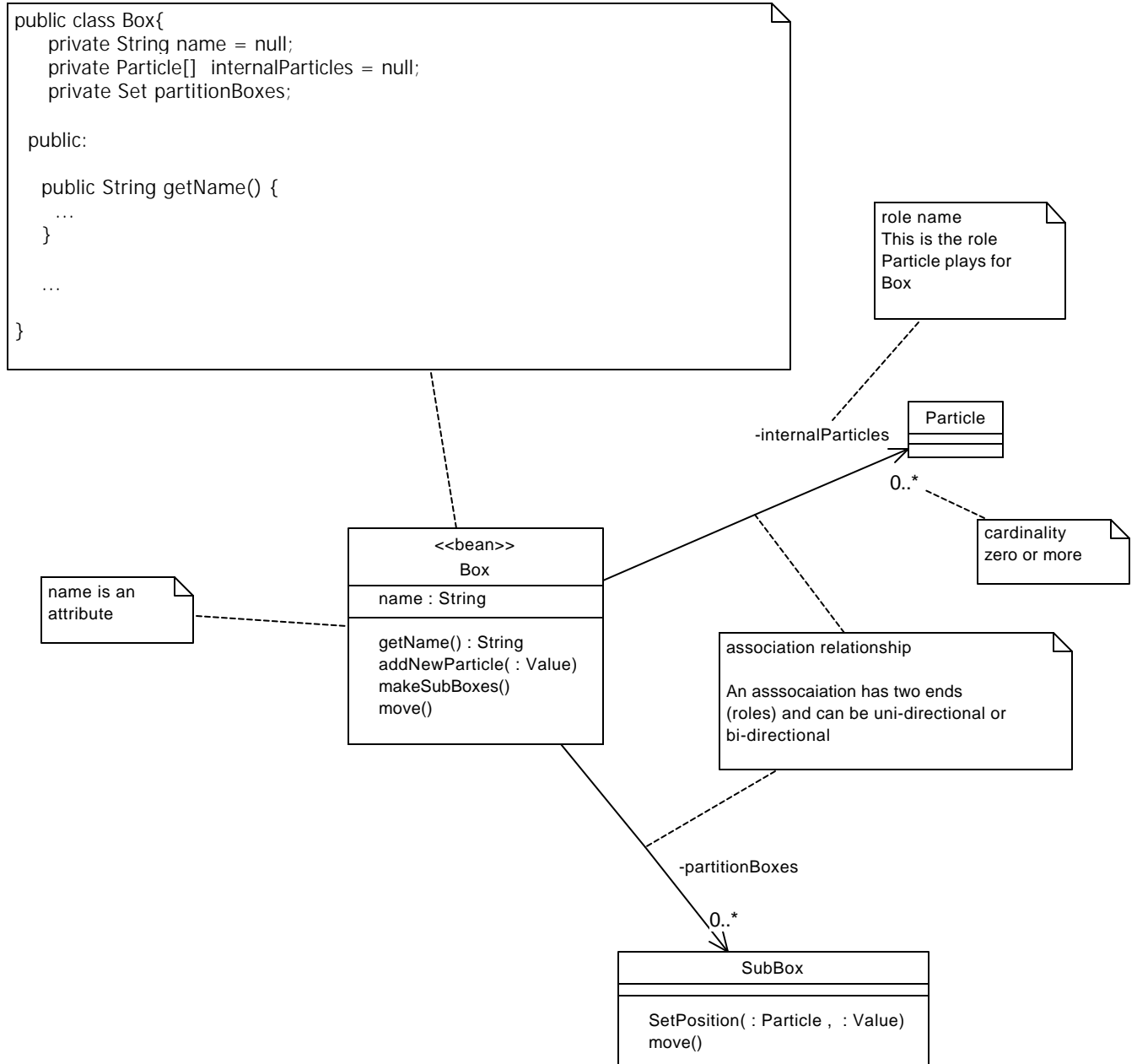
Class Diagram (a static view of the logical model)



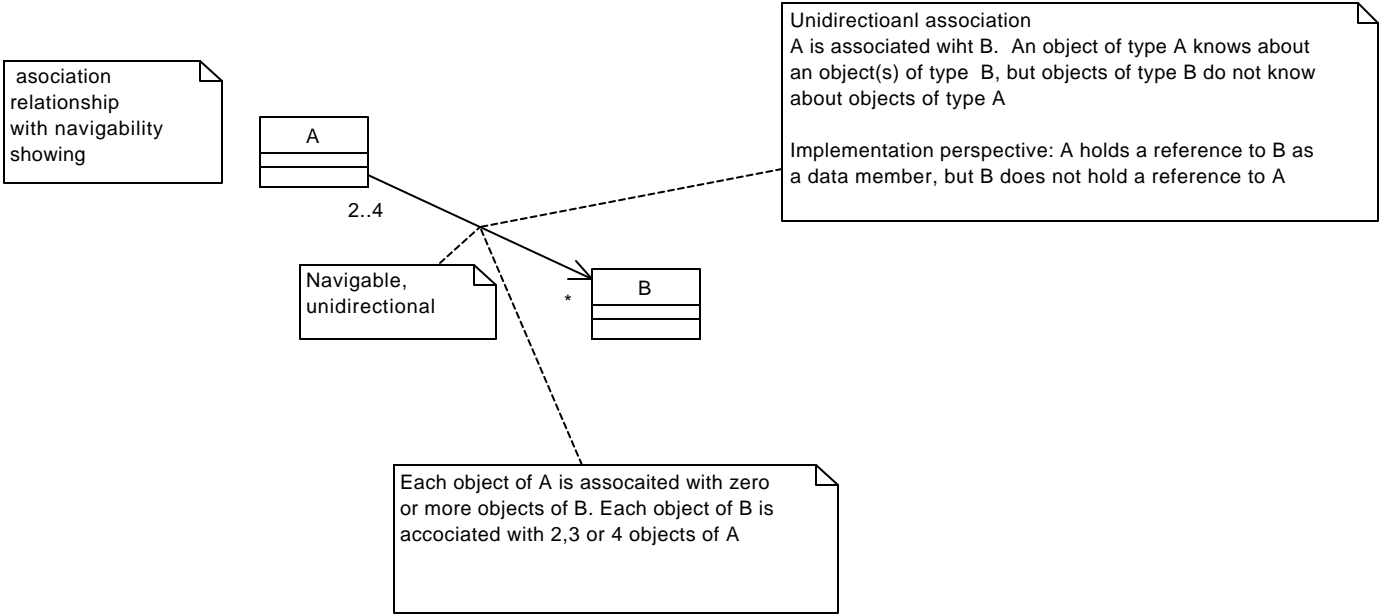
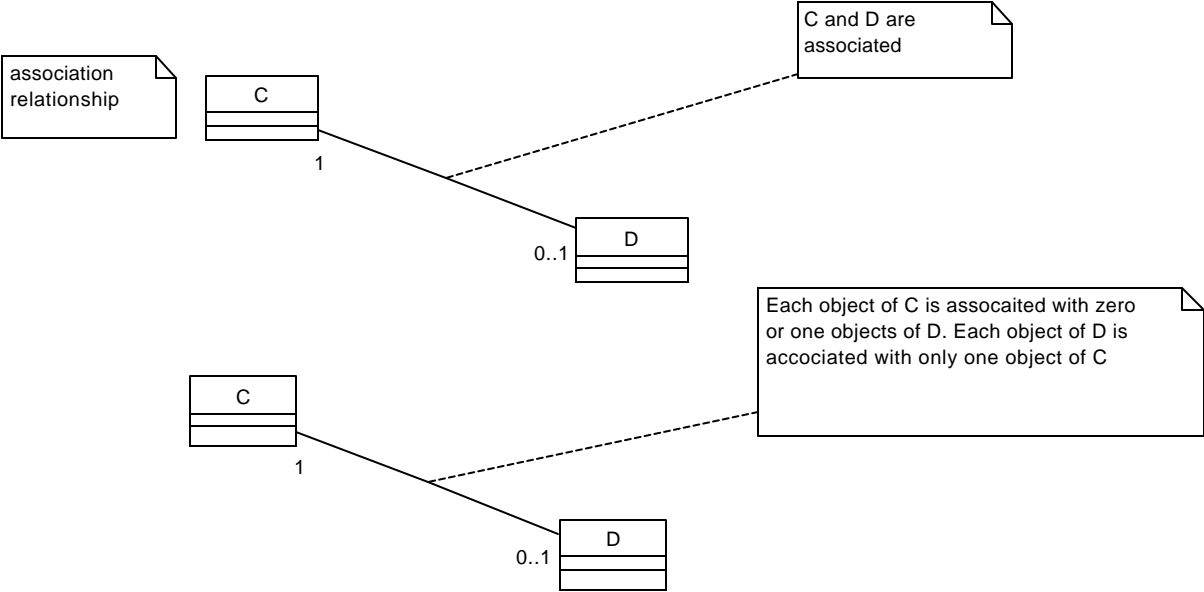
One need not show all the attributes, and signatures of the operations



Class Relationships (implementation perspective) (Static Class Diagram)

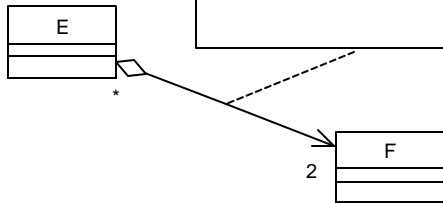


Association Relationships (relationships between instances of classes)



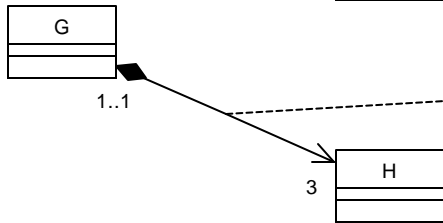
Association Relationships (relationships between instances of classes)

association relationship (aggregation)



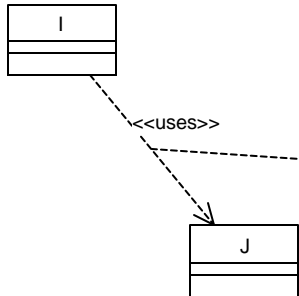
Aggregation (part-of relationship)
 Each object of E has exactly two objects of F as its part. A single object of type E does not necessarily own these two objects of type F. Other objects can also have these two objects of type F as their parts. An object of type F is a part of zero or more objects of type E
 Implementation perspective: E holds a reference to F as a data member. E does not necessarily control the life time of F

association relationship (composition)

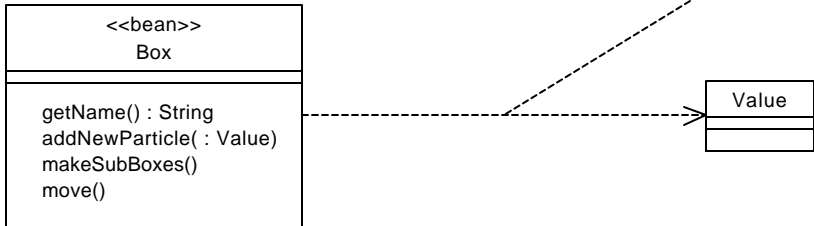


Composition (stronger aggregation)
 An object of type G owns three objects of type H. Each of these three objects of type H belongs to exactly this single object of type G. Objects of type H live and die with G.
 Implementation perspective: Only one object of type G holds a reference to three objects of type H as data members. Other objects do not hold references to these three objects of H. G controls the life time of H.

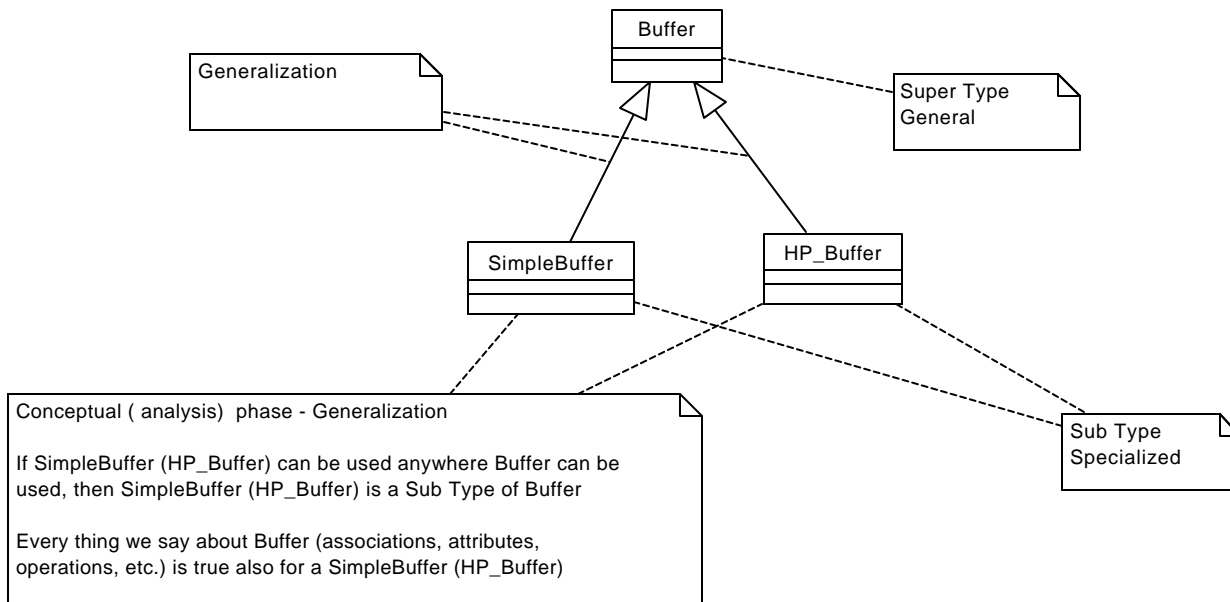
dependency relationship (use relationship)



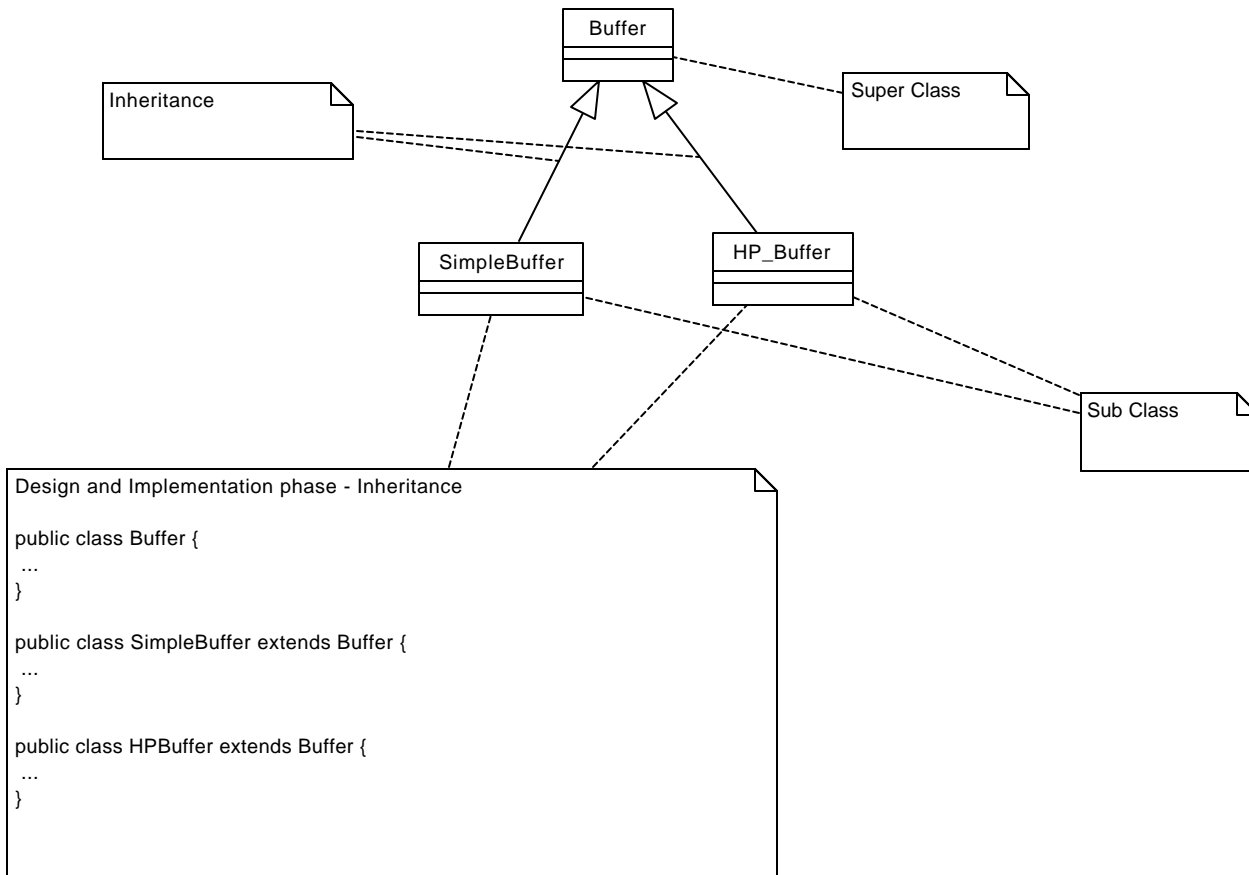
I uses J
 A change in J will effect I.
 Implementation perspective: An operation in I takes reference to an object of J as a parameter (as in example below), or uses a reference to object of J in its implementation.



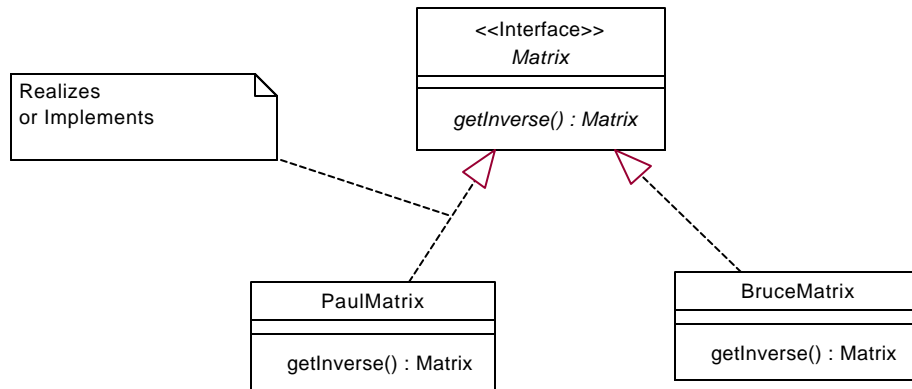
Generalization Relationship



Implementation of Generalization as Inheritance (not the only way to implement generalization)



Implements Relationship (design and implementation perspective)



Realizes or Implements relationship

Useful in the design and implementation phases.

Interface `Matrix` consists of the prototypes of the operations without implementation. `PaulMatrix`, and `BruceMatrix`, actually implement these operations.

If we declare a references variables to `Matrix`,
`Matrix aMatrix = null;`

Then either an object instantiated from `PaulMatrix`, or from `BruceMatrix`, can be plugged into this variables, and the right operation is called (polymorphism)

```
aMatrix = new PaulMatrix();
aMatrix.invertMatrix() invokes the method implementation in PaulMatrix
```

```
aMatrix = new BruceMatrix();
aMatrix.invertMatrix() invokes the method implementation in BruceMatrix
```

```
public interface Matrix {
    public Matrix getInverse();
}

public class PaulMatrix implements Matrix() {
    public Matrix getInverse() {
        // Paul's mplementation
    }
}

public class BruceMatrix implements Matrix() {
    public Matrix getInverse() {
        // Bruce's mplementation
    }
}
```