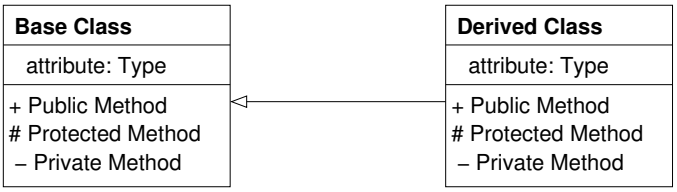
Part I.
Class Diagram

1. Relations

1.1. Generalization [◁——]



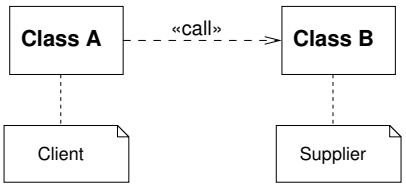
Properties:

- inheritance (derived class specialized from base class)

Implementation:

```
class Derived : public Base
{
...
}
```

1.2. Dependency [<- - -]

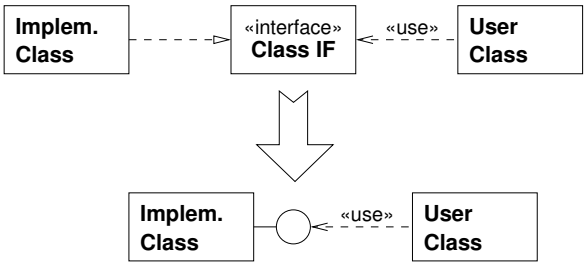


Properties:

- semantic dependency: changing supplier requires change of client
- four types: abstraction (interface), usage, permission, binding (template)

1.3. Realization [◁- - -]

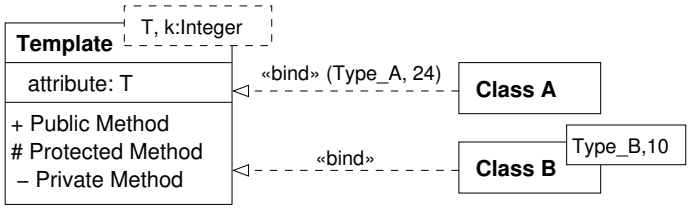
1.3.1. Interface



Properties:

- an interface is a specifier for externally-visible operations of a class, hiding any internal structure
- an interface has no implementation (and no attributes)
- in general a realization is a specialized abstraction relationship between implementation (client) and specification (supplier).

1.3.2. Template (Parameterized Class)

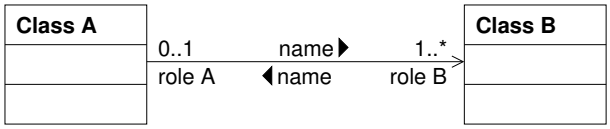


Properties:

- defines a family of classes
- cannot be instantiated
- «bind» actual parameters to formal parameter-list

1.4. Associations

1.4.1. General Association [<——]



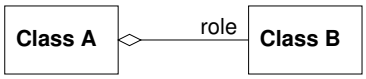
Properties:

- general relationship, with unspecified containment (cf. composition/aggregation)
- may be navigable — arrow indicates direction
- cardinality allows specification of n:m multiplicity relations

Implementation:

```
class A
{
    B* role_B;
}
```

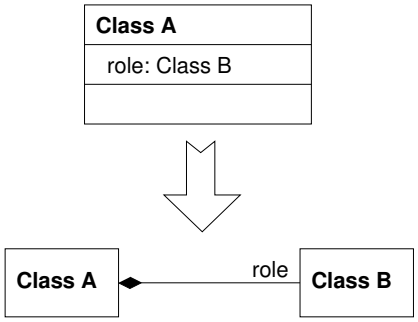
1.4.2. Aggregation [<◇]



Properties:

- ”has” relationship
- aggregate (B) lives independent of ”whole” (A)

1.4.3. Composition [<◆]



Properties:

- special form of aggregation
- ”contains” relationship
- ”part” (B) lives not longer than ”whole” (A)

Implementation:

```
class A
{
    B role;
}
```

2. Stereotypes

2.1. Classes

«auxiliary»	supporting class (for focus class)
«enumeration»	enum
«focus»	core class, supported by auxiliary class(es)
«implementationClass»	...
«interface»	not instantiable (abstract) class
«metaclass»	class whose instances are also classes
«primitive»	a primitive type/class
«struct»	C/C++ struct (only VC++)
«type»	...
«typedef»	C/C++ typedef (only VC++)
«union»	C/C++ union (only VC++)
«utility»	a class w/o instances

2.2. Relations

«refine»	refinement at different semantic levels (abstraction)
«derive»	derivation of client (by computation) from supplier (abstraction)
«trace»	used for tracking requirements and changes in models (abstraction)
«substitute»	denotes runtime substitutability not based on inheritance of structure (abstraction)
«call»	client invokes supplier operation (usage)
«instantiate»	client creates supplier instances[1] (usage)
«send»	source sends a signal to target (usage)
«responsibility»	client has some kind of obligation to supplier (usage)

2.3. Operations

2.4. Attributes

3. Constraints

[1] Equivalent to «create» in UML 1.x, which is (among others) obsolete.