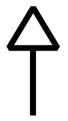
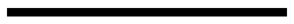
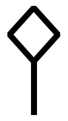
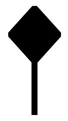
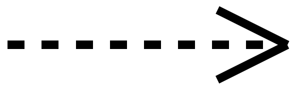
Class Relationship Summary

Categories **Properties**	**Inheritance** (Generalization)	**Association**	**Aggregation**	**Composition**	**Dependency** (Using, Delegation)
Semantics (meaning)	• *Is a* relationship • *A kind of* relationship • Subclass inherits attributes & operations from superclass	• *Has a* that reads well in both directions:	• *Has a* relationship • *Part of* relationship • Build a complex whole-object from simple part-objects	• *Has a* relationship • *Part of* relationship • Build a complex whole-object from simple part-objects	• One object *depends* on another object • One object *uses* the services of another • An object *delegates* some responsibility to another
Example	A student *is a* person.	A class *has a* teacher. A teacher *has a* class.	A car *has an* engine. An engine is *part of* a car.	A car *has an* engine. An engine is *part of* a car.	A calculator *depends on / delegates to* a listener. A button *uses* a listener.
Class Roles	Parent & Child Superclass & Subclass Base & Derived	Peer	Whole & Part	Whole & Part	Dependent/Independent Client/Server User/Supplier
Directionality (Navigation/Knowledge)	Unidirectional (Child to Parent)	Bidirectional	Unidirectional (Whole to Part)	Unidirectional (Whole to Part)	Unidirectional (Client to Supplier)
Object Binding	Strong	Weak	Weak	Strong	Temporary or Transient
Lifetimes	Coincident	Independent	Independent	Coincident	Independent
Sharing	Exclusive	Shareable	Shareable	Exclusive	Shared
Implementation	`: public`	2 pointer variables	1 pointer variable	1 non-pointer variable	1 local variable
Variable(s)	N/A	Class scope both classes	Class scope whole class	Class scope whole class	Client function local var

Code Pattern	`class A {};`
	
	`class B : public A`
	`{`
	`    ...`
	`};`
		`class B`
		`{  A* a;  };`
		
		`class A`
		`{  B* b;  };`
			`class B {};`
			
			`class A`
			`{`
			` B* b;`
			`};`
				`class B {};`
				
				`class A`
				`{`
				` B b;`
				`};`
					`class A`
					`{`
					`public:`
					`  __func(B b) {}`
					`  __func(B* b) {}`
					`  __func(B& b) {}`
					`};`

UML Class Relationship Symbol					

Relationship Table Legend: Properties and Values

Property	Values		Meaning
Semantics[1]	Is a, Kind of		One object is another object (possible to substitute one object for another). One object is a (special) kind of another (general) object.
	Has a, Part of		One object has another object as a part. One object is a part of another object. One object contains another object.
	Depends on		One object depends on (uses, delegates some responsibility to) another object.
Directionality[2]	Bidirectional		Messages are exchanged in both directions (i.e., both objects may send a message). Possible to go from either object to the other. Both objects "know" about each other.
	Unidirectional		Messages are sent in only one direction (i.e., only one object sends a message). Possible to go from one object to another but not in the opposite direction. Only one object "knows" about the other.
Binding[3]	Strong/Tight	Lifetime	(Coincident) Object lifetimes are the same: they are created and destroyed at the same time
		Sharing	(Exclusive) Objects form an exclusive relationship that does not permit sharing of the parent or part object
	Weak/Loose	Lifetime	(Independent) Object lifetimes *may* be different: they may be created and destroyed when convenient (at the same or at different times).
		Sharing	(Shareable) Whole objects may share their pars with other objects in the program/design.
	Temporary/ Transient	Lifetime	(Independent) Object lifetimes are different: The client passes an existing object (itself or one of its parts) as an argument to a supplier function's parameter, which the function destroys when it returns.
		Sharing	(Shareable) Parameters passed by pointer or reference are shared; parameters passed by value or non-parameter, local variables are not.

1. What the relationship means in the problem domain (i.e., the “real world”).
2. The direction that messages are exchanged between objects. Given one object in the relationship, is it possible to navigate to or access the other object.
3. Binding strength summarizes two properties: First, the lifetimes of related objects are the same (or coincident) if the program creates them at the same time, establishes the relationship then, and destroys them at the same time; otherwise, the lifetimes are different (or independent). Second, the part object is shareable if two or more objects can "have it" simultaneously; otherwise, it is held exclusively by one object.