Everything on Implementation Issues

The V model
A view of the relationship between analysis design coding and testing.

You test each component individually and group components to see i they work together

Detailed design activities

Reviewing/Refining	Adding/removing classes
	Reviewing attributes and operations
	Reviewing associations
Refactoring	Altering the internal design of the model without altering its visible behaviour
	Refactoring is about improving the overall model, it can be applied to code as well
	Improves manageability, increases reuse/optimisation, fits to architectural framework
Selecting a framework	A software architecture defines the system decomposition, global control flow patterns and communication protocols.
	A standard framework can make design and implementation easier, eg MVC.
Concretizing Associations	Fixing details and choosing an implementation
Implementing the classes	Basically coding and introducing private attributes and operations

An association represents a conceptual relationship between instances of classes

Refining the association

Identify if its a dependency

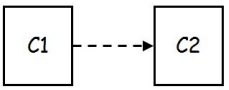
Decide if its a conceptual aggregation or composition

Identify its navigability/directionality

Identify its multiplicity

Dependency

C1 depends on C2, if C1 has an operation parameter of type C2 or local variable of type C2.
Changes in C2 might require changes in C1s code



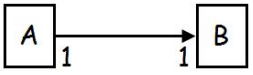
If an association is not simply a dependency C1 can have persistent ownership of, or access to C2 and C1 may call public operations of C2

The association will be navigable from client C1 to supplier C2

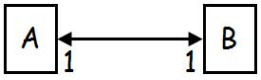
To implement this C1 will have an attribute holding an instance of C2. In Java that's a global/instance variable holding a reference to a C2

Implementing Associations

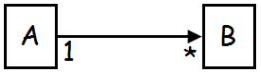
1-1 : Association unidirectional navigability from A to B. A has an attribute (instance variable) holding a reference to an instance of B. The attribute needs to be initialized.



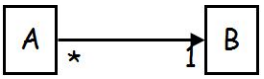
1-1 : Association bidirectional navigability from A to B. Each class has an attribute holding a reference to the other. Setting up the references is similar to unidirectional case.



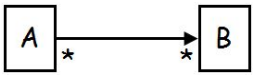
1-Many : Navigable from 1 to the many. A must have an attribute that is a collection of references to instances of B. eg array, vector, hashtable. Many not represented in code



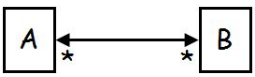
1-Many : Navigable from Many to the 1. Same as 1-1. A simply needs an attribute private B theB



Many-Many : Navigable in one direction only. Same as 1 to many, the * is for design purposes.



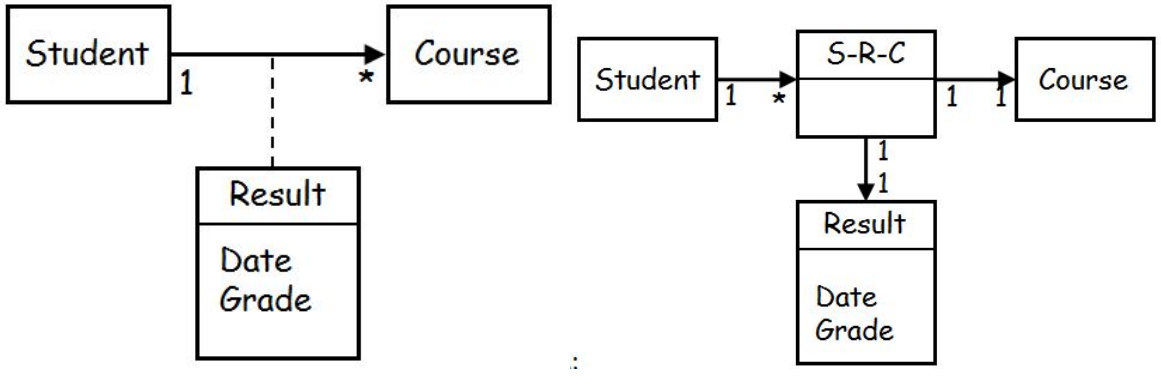
Many-Many : Navigable in both directions. Both A and B must have attributes that hold a collection of references to the other



An alternative approach for implementing 1-Many navigable in one direction is to introduce an intermediary collection class

Implementing Association Class

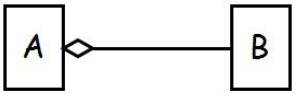
An association class is information attached to an association. An implementation is to add another fourth class, linking all three classes together. This version is not necessarily good decision



Aggregation

Aggregation indicates a conceptual/hierarchical structure, a whole-part relationship between separate objects. Often 1-1 or 1-Many from A to B.Navigability is usually 1 way. The design is indicating that although collection within A, the object B has a separate existence from A. Can be created outside A, or maybe passed to A as parameters or returned by A as results. Other objects may have references to them.

A must hold references to B to enable this



Composition

Composition is a strong form of aggregation. The design indicates the parts have no separate conceptual existence from the whole. Objects of class B are created and destroyed by A. Each instance of B belongs to one and only one instance of A. A never gives reference to its Bs.