



**DOMAIN CLASS DIAGRAM MODELING
STANDARDS AND GUIDELINES
VERSION 1.0**

DECEMBER 2, 2005

Information and Technology Management Branch

❖ IM / IT Standards & Guidelines ❖

Table of contents

DOCUMENT PURPOSE	3
DIAGRAM PURPOSE	3
DIAGRAM ELEMENTS	3
CLASS.....	3
ASSOCIATION	4
ASSOCIATION PROPERTIES.....	6
ASSOCIATION CLASS	7
NAMING STANDARDS	7
CLASS NAMING	7
CLASS ATTRIBUTE NAMING	7
CLASS OPERATION NAMING	8
ASSOCIATION NAMING	8
MODELING STANDARDS	8
DOMAIN CLASS DIAGRAM MODELING.....	8
CLASS MODELING	9
CLASS ATTRIBUTE MODELING	9
CLASS OPERATION MODELING.....	9
ASSOCIATION MODELING	9
SPECIFICATION STANDARDS	10

Document Purpose

The purpose of this document is to provide a set of minimal acceptable standards and guidelines for modeling the *Domain Class Diagrams* based on UML 2.0. The document briefly describes the elements of a typical *Domain Class Diagram* followed by naming and modeling standards for the elements. The document assumes that the reader has prior knowledge of UML and Class Diagram modeling.

This document does not provide standards for the persistent classes and standards for the detailed design-level classes such as User interface classes, Controller/process classes etc. Some standards and guidelines on such classes are available at <http://www.msd.gov.bc.ca/itmb/destandards/downloads/javamodelguide.pdf>.

Diagram Purpose

UML class diagram is used to model the structural aspects of the system; i.e. classes and its attributes (fields) and operations (methods) and also the interrelationships (association, composition, aggregation and generalization) between the classes. Class diagrams are used for domain modeling as well as for detailed design modeling. This document describes the standards for *domain modeling*. Domain modeling is an inside-out approach, meaning the modeling starts with the identification of core classes in the system and then working from inside outward to see how the classes participate in the system. Domain model is a representation of real world conceptual classes, not of software components.

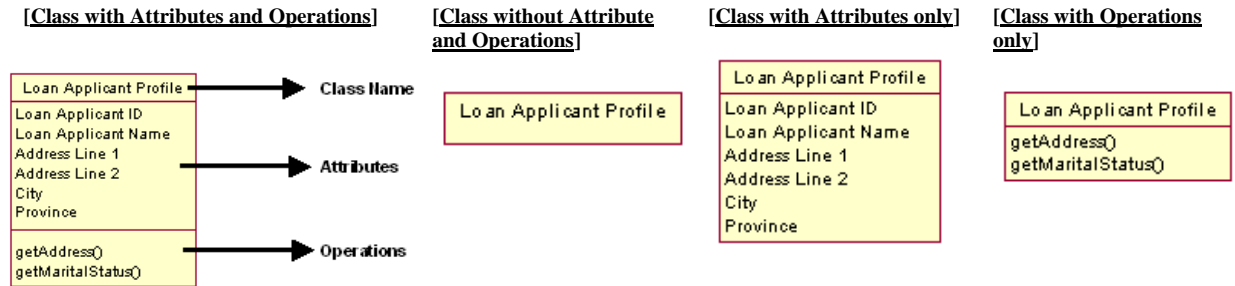
Diagram elements

The Class Diagram is comprised of the following model elements.

Class

Classes form the main building blocks of an object-oriented application. A *class* is a model element that represents an abstraction of a concept or thing. It identifies the attributes, operations, relationships, and semantics that objects created (instantiated) from the class will inherit.

The *Class* is drawn as a rectangular box with three compartments as shown in the examples below. The upper compartment contains the *class name*. The middle compartment contains the *attributes* and their details, which represents the information or data of the class. The lower compartment contains the *operations* and their details, which represents the behavior of the class. Most UML tools provide the facility to hide the middle and lower compartments to enable abstractions, as shown in the examples below.



Association

Objects are often associated with, or related to, other objects in a uni-directional or bi-directional manner. In a Domain Class Diagram an *Association* is a relationship between two classes whose instances (objects) are involved in the relationship.

A *bi-directional association* line is drawn with no shapes attached to its ends (\longleftrightarrow). A *uni-directional association* line is drawn with an arrowhead at one end (\rightarrow) and an optional diamond shape at the other end to depict aggregation or composition respectively ($\diamond\rightarrow$ or $\blacklozenge\rightarrow$).

Associations may have other properties such as name/label, role and multiplicity, which are discussed in later sections.

There are four types of associations that can exist between classes :

Simple Association : This relationship specifies a non-aggregate and non-composite simple association between two classes. A *Simple Association* may be *unidirectional* or *bi-directional*.

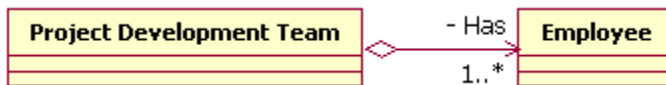
Uni-directional simple association between two classes is used when the second class need not know about the first class and need not pass messages to the first class at run time. A *uni-directional simple association* is drawn as a solid line with an arrowhead at one end as shown in the example below.



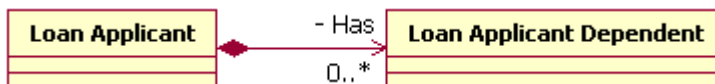
Bi-directional simple association between two classes is used when the two classes need to know about each other and need to pass messages between each other at run time. A *bi-directional simple association* is drawn as a solid line without any arrowhead as shown in the examples below.



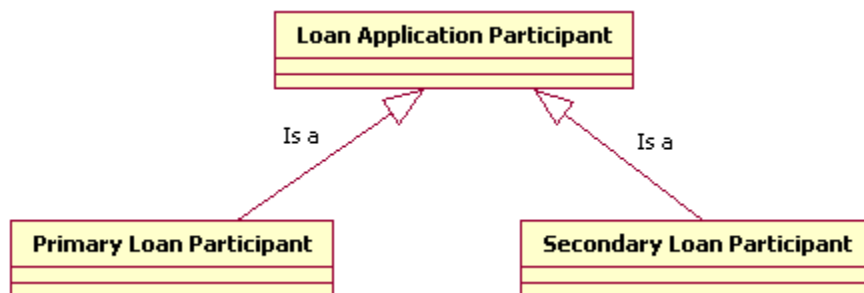
Aggregation Association : This relationship (which is also called "weak aggregation" and is a specialization of simple association) specifies the "whole-part" relationship between two classes. Aggregation association is modeled when there is a need for the "part" class to exist independent of the aggregate. Aggregation association is drawn as a solid line with an unfilled diamond shape at the "whole" side and an arrowhead at the "part" side of the association (◊→) as shown in the examples below.



Composition Association : This relationship (which is also called "strong aggregation") specifies the "whole-part" relationship between two classes where one class has life-time dependency on the other. Composition association is drawn as a solid line with a filled diamond shape at the "whole" side and an arrowhead at the "part" side of the association (◆→) as shown in the examples below.



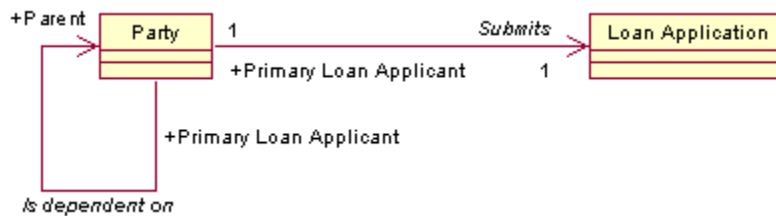
Generalization (inheritance) Association : Generalization (inheritance) association is used for modeling the "is a" and "is like" relationships. Generalization association between two classes is drawn as a solid line with a closed arrowhead pointing towards the parent class (→) as shown in the examples below.



Association Properties

Association Name/Label : Association name is typically a one or two-words short description of the association. Association name is optional. A unidirectional association can have two different names/labels - one each attached to its two ends. All other associations have their single name/label typically attached at the center or end of the association. See the earlier examples, where the association name/label is shown in the diagrams.

Association Role : A role indicates the context that the class takes with regard to the association. Association Role is optional. Roles are typically used when the association does not have a name/label attached, or it is not clear from the association name/label what the roles are, or if there is a recursive association, or if there are several associations between two classes. Roles are typically not used when a self-explanatory name/label is already present for the association. See the example of roles below.

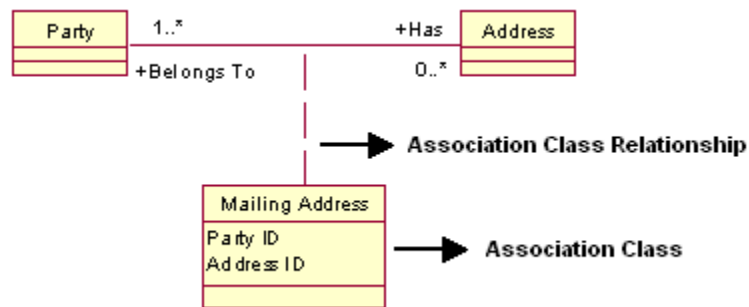


Association multiplicity : Multiplicity in an association indicates the number of objects (instances) of a class that can participate in the relationship with a single object (instance) of another class. Multiplicity is usually specified as a single integer or an integer range (two integer values separated by two dots). Multiplicity may also be specified as any specific value. Most UML tools provide following standards for specifying multiplicity in the associations :

Multiplicity Indicator	Meaning
0..1	Zero or one
1	One only
0..*	Zero or more
1..*	One or more
n	Only n (where $n > 1$)
0..n	Zero to n (where $n > 1$)
1..n	One to n (where $n > 1$)

Association Class

Association class (also known as "link class") is used to model the associations that have attributes and operations. If there are attributes and operations that cannot be modeled in either of the classes in a relationship but are shared by both classes, then those shared attributes and operations are modeled in a special class called "*association class*". The "*association class*" is connected to the original two classes' association through a special relationship called "*association class relationship*", which is drawn as a broken line without arrowhead and without any name/label as shown in the example below.



Naming standards

Class Naming

- Name of the class should be a singular noun.
- Name of the class should reflect the role it plays in the system.
- Name of the class should be picked from the business glossary of the system.
- Use intuitive names.
- For names with more than one word such as "Primary Loan Applicant", each word should begin with upper case and words should be separated by a space. Prepositions if any (such as "in", "to", "from") should be completely in lower case.

Class Attribute Naming

- *Class attribute* should be named as a domain-based noun or a noun phrase or a possessive phrase.

Class Operation Naming

- ***Avoid defining of operations for classes in the domain class diagram altogether*** as there would not be enough information available in the analysis stage to derive operations.
- *Class operations*, if modeled in the domain class diagram for some reasons, should be named as a verb.

Association Naming

- *Association* should be named in active voice.
- Keep the association names concise and do not use long names.
- Specify roles for associations only in the following cases :
 - The Association does not have a self-explanatory name or label.
 - It is not clear from the association name/label what the roles are.
 - If there is a recursive association.
 - If there are several associations between two classes.
- Use bi-directional association only when collaboration between two classes is needed in both directions.
- Use Generalization (inheritance) association sparingly.

Modeling standards

Domain Class diagram modeling

- Keep adequate levels of abstraction. Do not do exhaustive noun and verb analysis to identify classes, attributes and operations.
- ***Avoid defining of operations for classes in the domain class diagram altogether*** as there would not be enough information available in the analysis stage to derive operations. However, if for some reason operations are to be defined, then follow a minimalist approach while defining operations for domain classes.
- Use simple aggregation during domain modeling.
- Do not debate over issues like "aggregation versus composition" since these are detailed design issues.
- Do not model implementation specific things in domain model. Focus on problem domain.
- Do not create one-to-one mapping between domain classes and relational tables since a relational table can have a lot of attributes that might not belong together in the context of an object model.

Class modeling

- Ensure that classes in the domain class diagram accurately represent the real abstractions that the problem domain presents.
- The best sources of *domain classes* are likely to be the Enterprise Data Model (EDM), high-level problem statement and expert knowledge of the problem space.
- Always use box shape for drawing classes and avoid using stereotypes and other shapes.

Class Attribute modeling

- In the Domain Class diagram, do not model design issues of the attributes such as their visibility (public, private, protected).

Class Operation modeling

- ***Avoid defining of operations for domain classes altogether*** as there would not be enough information available in the analysis stage to derive operations. However, if for some reason operations are to be defined, then follow a minimalist approach while defining operations for domain classes.

Association modeling

- Center the "association class relationship" while connecting it to the association between the original classes.
- Model associations horizontally as much as possible.
- Avoid modeling of just "*" alone as a multiplicity - always precede the "*" with 0 or 1 or n (where n > 1). Examples : 0.* ; 1.* ; n.*.
- Avoid diagramming cluttering and do not model implied relationships.
- Do not model every single dependency.

Specification standards

Domain Class Diagram has no specification or template explicitly for documentation purposes. However, documenting the Domain Class model is very important. Most UML tools provide in-built documentation capturing and printing capabilities for the Class diagram and its elements. Use this facility to sufficiently document the class diagram and its elements.