Class Diagram

Elements in Class Diagram:

A class diagram is composed primarily of the following elements that represent the system's business entities:

- ***Class:*** A class represents an entity of a given system that provides an encapsulated implementation of certain functionality of a given entity. These are exposed by the class to other classes as **methods**. Apart from business functionality, a class also has properties that reflect unique features of a class. The properties of a class are called as **attributes**.
  The UML representation of a class is a rectangle containing three compartments stacked vertically as shown in figure 5.1.

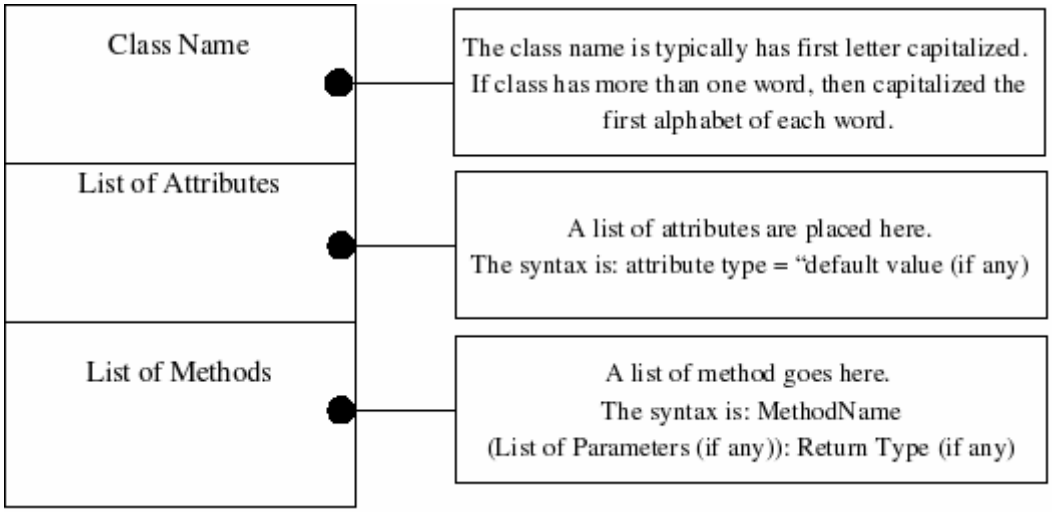

Figure 5.1 The Structure of a Class

The top compartment shows the class name. The middle compartment list the class's attributes. The bottom compartment lists the class's operations. When drawing a class element in class diagram, you must use the top compartment, and bottom two compartments are optional.
Figure 5.2 shows a circle modeled as class.

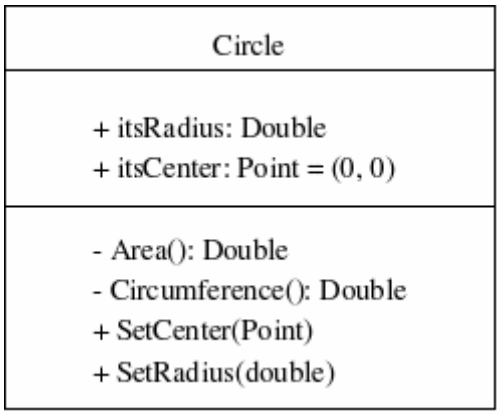

Figure 5.2 A Circle class showing name, attributes, and methods

Class Attributes List

The attribute section of a class lists each of the class's attributes on a separate line. The attribute section is optional, but when is used it contains attribute of the class displayed in a list format. The syntax is:

name: attribute type

In business class diagrams, the attribute types usually correspond to units that make sense to the likely readers of the diagram (i. e., minutes, cm, etc.). However, a class diagram that will be used to generate code needs classes whose attribute types are limited to the types provided by the programming language, or types included in the model that will also be implemented in the system.

Sometimes it is useful to show on a class diagram that a particular attribute has a default value. The attributes with default value can be shown as below:

name: attribute type = default value

For e. g., itsCenter: Point = (0, 0)
Showing a default value is optional.

Class Operations List

The class's operations are documented in the third compartment of the class diagram. Like attributes, the operations of a class are displayed in a list format, with each operation on its own line. Operations are documented using following syntax:

name(parameter list): type of return value

Access Modifier

You can also apply access modifiers such as public access, protected access, private access, and package access applied to methods and attributes of a class, if required. These access modifiers determine the scope of visibility of the attributes and methods of a class. The notations used for indicating visibility are as shown in figure 5.3.
You can also add documentation information to a class. Notes and constraints can be added to a list of attributes. Notes contain additional information for reference while developing the system, whereas constraints are business rules that the class must follow, and are text included in curly brackets.

Public	+
Protected	#
Private	-
Package	~

Figure 5.3: Access Modifiers

During the early phase of the system design conception, classes called Analysis classes are created. Analysis classes are of the following types as per their behavior:
Boundary Class: In an ideal multi tier system, the user interacts with system only with the classes called boundary class. For example, JSPs in a a typical MVC architecture form a boundary class.
Control Class: These classes typically don't contain any business functionality. However, their main task is to transfer control to the appropriate business logic class, depending on the inputs received from the boundary classes.
Entity Class: These classes are those that contain the business functionality. Any interactions with back-end systems are generally done through these classes.

Figure 5.5 Representation of Analysis Classes

- ***Interface:*** An interface is a variation of a class. A class provides an encapsulated implementation of certain business functionality of a system. An interface on the other hand provides only a definition of business functionality of a system. A separate class implements the actual business functionality.
  So, why would a class not suffice? You can define an abstract class that declares business functionality as abstract methods. A child can provide the actual implementation of the business functionality. The problem with such approach is that your design elements get tied together in a hierarchy of classes. So, even though you may not have intended to connect your design elements representing drastically different business entities, that is what might result. Hence, the use of interface design construct in class diagram. Different classes belonging to different and discrete hierarchies can maintain their distinct hierarchies and still realize the functionality defined in the methods of the interface.
  An interface shares the same features as a class; in other words, it contains attributes and methods. The only difference is that the methods are only declared in the interface and will be implemented by a class implementing the interface. Thus, a class can have an actual instance of its type, whereas an interface must have at least one class to implement it.
  The interface is represented same as class and stereotype <<interface>> is added to indicate interface. For example, consider figure 4.5 in which both teacher and student classes implement the Person interface and do not inherit from it.

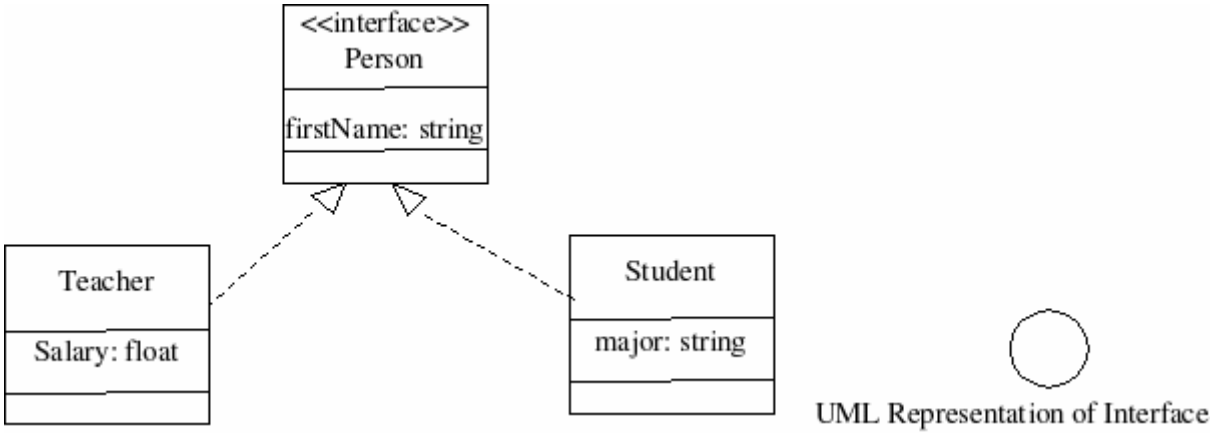

Figure 5.5 Example of an interface

- ***Package:*** Packages are general-purpose grouping mechanisms. A package is purely conceptual. A package provides the ability to group together classes, interfaces, components, nodes, collaborations, use cases and even other packages which are either similar in nature or related. Grouping these design elements in a package element provides better readability of class diagrams, especially complex class diagrams.
  Packages have a name that uniquely identifies the package. A package is represented as a tabbed folder, usually including only its name and, sometimes, its contents as shown in figure 5.6.
  For class package diagram, you can apply following thumb rules: First, classes in the same inheritance hierarchy typically belong in the same package. Second, classes related to one another via composition often belong in the same package. Third, classes that collaborate with each other a lot often belong in the same package.

Figure 5.6 A package

Previous Content Next