Introduction to Class Diagrams

Posted by potty [1] on January 22, 2014 at 10:45 PM CST
ide_icon.png [2]

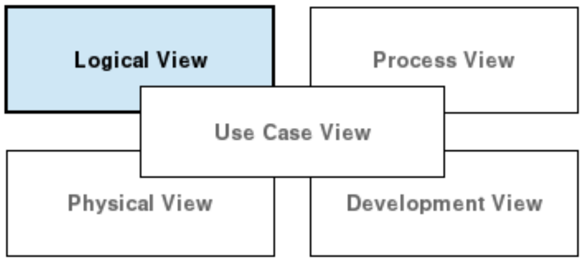
In this blog post you will learn to prototype your applications using UML before typing a line of code. This is useful because you can generate a code template from these diagrams.

Introduction to Class Diagram

In any project's planning phase, the programmer should describe the behaviour of the different objects that are required to satisfy business use cases. However objects can't emerge from nothing, so you need a template that describe the type of objects that the system will have. As a result, you will need to show a diagram with all the classes involved and their relationships. This diagram is called Class Diagram.

According to Wikipedia, a Class Diagram "in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects". Although this definition is pretty clear, you have to understand that Class Diagram are part of the system model's Logical View.

On the following image you will see its relation with the other project views:



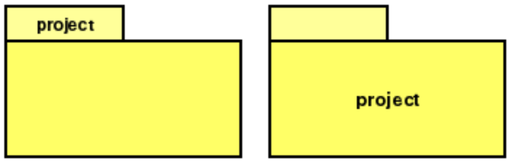
Let's Remember: OOP Basic Concepts

A class is a blueprint from which objects are generated. It includes the attributes and methods of that object. Now, in the real-life, we can find many individual objects of the same type. Let's say a car. There may be a lot of cars, but all with the same basic components. In this case, the concept of the car is the class (that can belong to a package) and the many cars in the world are instances of that class. This analogy applies to almost anything.

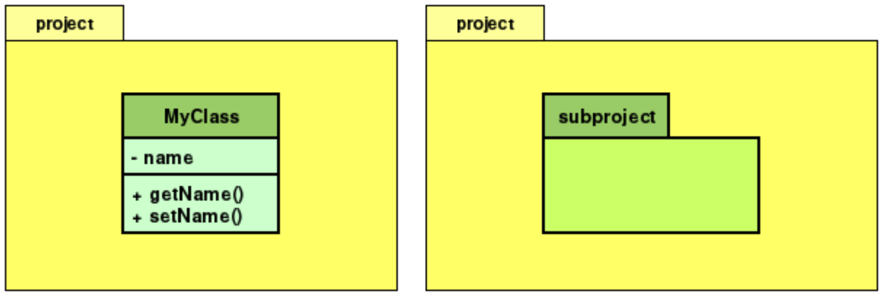
If you like to learn more about object-oriented programming concepts, I invite you to read my previous post [3].

Representing Packages in UML

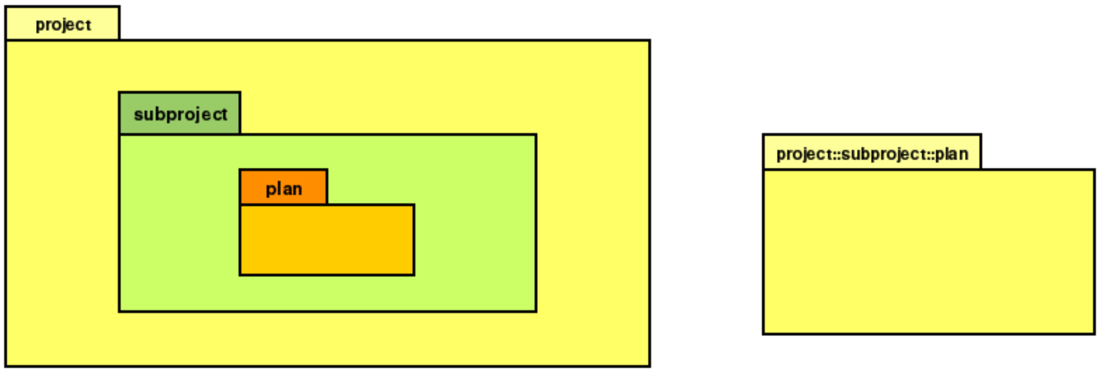
In UML, a package is drawn by a folder with a tab. The name of the package is written inside the folder or in the tab. See this on the following diagram:



A package organize UML elements like classes and other packages. The following diagrams show the proper way to represent them:



Finally, it's common to see a deeply nested packages. For this case, you can use an alternate notation that can ease the work. You can represent the previous diagram with the alternate notation as follows:

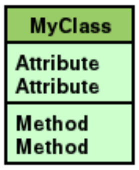


Recommendation: The name of the package should be in lower case. In small projects it should be simple and meaningful, but in complex projects the name should be subdivided and specific.

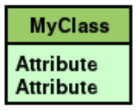
Representing Classes in UML

In UML, a class is drawn by a three sections rectangle. The top section includes the name of the class, the middle section include the attributes and the bottom section include the methods. Moreover, there are four different ways of showing classes using the UML notation:

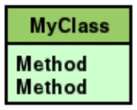
- Class Name + Attributes + Methods



- Class Name + Attributes



- Class Name + Methods

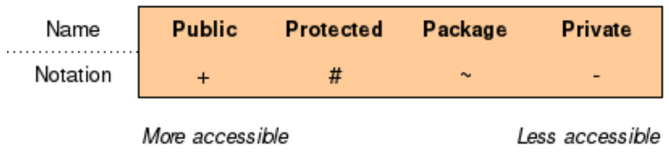


- Class Name

The name of the class should follow the CamelCase pattern. Also, try to use nouns because it represents a real-life object.

Visibility

To apply encapsulation, you have to limit the access to the attributes, methods and classes. This is achieved through visibility. There are four types of visibility. The following diagram represents an overview of them:



Public

Public the weakest visibility characteristic. Its notation is the plus symbol (+). Declaring a class attribute or method as public will make it accessible directly by any class.

Private

Private is the strongest visibility characteristic. Its notation is the minus symbol (-). Declaring a class attribute or method as private will make it only accessible for the class itself.

Protected

Protected is one the neutral visibility characteristic. Its notation is the hash symbol (#). Declaring a class attribute or method as protected will make it more visible than private attributes and methods, but less visible than public ones. In other words, protected elements can be accessed by a class that inherits from your class whether it is in the same package or not.

Package

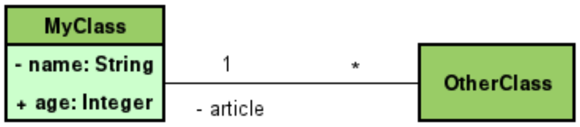
Package is the other neutral visibility characteristic. Its notation is the tilde symbol (~). Declaring a class attribute or method as package will make it visible only to any class in the same package. It doesn't matter if other class from other packages inherits from a class within the main package.

Recommendations

- Private is the most useful when you have an attribute or method that you want it to be independent from the whole system.
- Attributes should always be private and only in extreme cases be protected, never be public.
- Protected is crucial if you want allow access to an attribute or method in the base class without exposing the attribute or method to the whole system.
- If you want to reuse methods between classes but not expose it to the whole system then you will need the Package visibiilty.
- Protected and Package visibilities are not the same.

Attributes

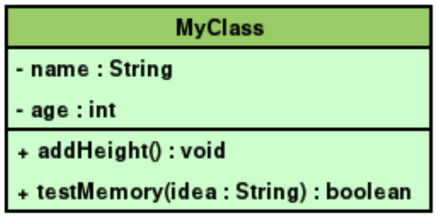
An attribute can be represented on a class diagram by placing them inside the middle section of the class box or by association with another class. It is important that the attribute has its visibility characteristic, proper name and data type. See the following diagram for more details:



The name of the attribute should be in mixed case. The names should represent the value of what it represents. Avoid using short names.

Methods

A method can be represented on a class diagram by placing them inside the bottom section of the class box. It is important that the method has its visibility characteristic, proper name, parentheses for parameters and return type. See the following diagram for more details:



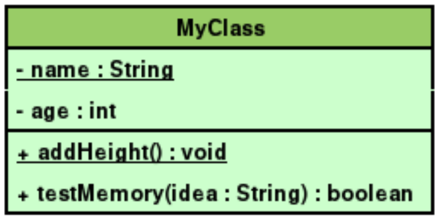
The parameters are used to specify the required input to allow the method to complete its job. Its notation is the following: `name : datatype`. If you are not using the `void` return type, you have to specify at least one parameter. In the scenario of multiple parameters, you should separate them with a comma.

The return type is specified after a colon at the end of the operation's signature. The name of the method should be in mixed case. Use verbs to describe what the method does.

Representing Static Elements in UML

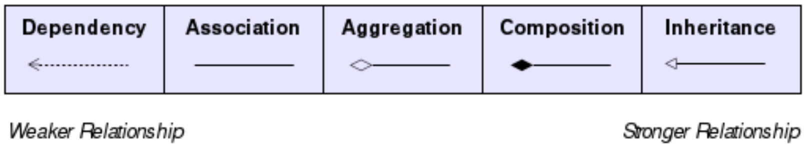
In object-oriented programming the non-static attributes and methods are associated with each individual object that is created from a class. In the other hand, static attributes and methods are associated with the class. This allows to shared its values among all the objects of that class type.

An attribute or method is made static in UML by underlining it. See this on the following diagram:



Representing Class Relationships in UML

It is a fact that classes coexist and work together through different releationships. The strength of a class relationship depends on how involved is each class with the other: tightly coupled or loosely coupled. The following diagram shows the five different type of class relationships:



Dependency

Declares that a class needs to know about another class to use objects of that class. Its representation is a dashed arrow.

Association

Means that a class will contain a reference to an object of another class in the form of an attribute. Its representation is the straight line, the association name and the attributes (or Class) involved. An example is "Student --take--> Course".

Aggregation

Indicate that a class owns but may share objects of another class. Its representation is by an empty diamond, a straight line, the aggregation name and the attribute shared. An example is "Person ?---- Address".

Composition

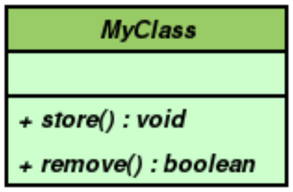
It work similar as Aggregation but in a stronger way. Its representation is by a filled diamond, a straight line, the aggregation name and the attribute shared. An example is "Person -----? Name".

Generalization

It is also called Inheritance. Used to describe a class that is a type of another class. Its representation is by an empty triangle and a straight line. An example is "Mammal -----? Dog".

Representing Abstract Classes in UML

An abstract class is a class that is extend by other classes. Unlike concrete classes, abstract classes can't be instantiated and only have methods. The main purpose of it is to have common code to use in subclasses. Its notation is similar as the Class but without attributes and the text is in italics.



Representing Interfaces in UML

An interface is a collection of operations that do not have corresponding implementation methods. Are much safer to use because they avoid the problems related to multiple inheritance. Think of it llike a contract.

In UML, an interface is drawn by a two sections rectangle. The top section includes the name of the interface with a guillemet (

You have no comments to approve.

Attachment	Size
image_1.png [4]	7.55 KB
image_2.png [5]	2.55 KB
image_3.png [6]	7.63 KB
image_4.png [7]	6.23 KB
image_5.png [8]	2.82 KB
image_6.png [9]	2.15 KB
image_7.png [10]	2.15 KB
image_8.png [11]	1.25 KB
image_9.png [12]	5.99 KB

image_10.png [13]	5.54 KB
image_11.png [14]	6.03 KB
image_12.png [15]	6.12 KB
image_13.png [16]	7.5 KB
image_14.png [17]	4.32 KB
image_15.png [18]	7.71 KB

Related Topics >> Open Source [19] Programming [20] Research [21] Java User Groups [22] Java Desktop [23] Community [24] Global Education and Learning [25] J2SE [26] Blogs [27]

Source URL: http://www.java.net/blog/potty/archive/2014/01/22/introduction-class-diagrams

Links:
[1] http://www.java.net/blog/potty
[2] http://www.java.net/sites/default/files/potty/ide_icon_0.png
[3] https://weblogs.java.net/blog/potty/archive/2014/01/20/introduction-object-oriented-programming-oop-part-i
[4] http://www.java.net/sites/default/files/image_1.png
[5] http://www.java.net/sites/default/files/image_2.png
[6] http://www.java.net/sites/default/files/image_3.png
[7] http://www.java.net/sites/default/files/image_4.png
[8] http://www.java.net/sites/default/files/image_5.png
[9] http://www.java.net/sites/default/files/image_6.png
[10] http://www.java.net/sites/default/files/image_7.png
[11] http://www.java.net/sites/default/files/image_8.png
[12] http://www.java.net/sites/default/files/image_9.png
[13] http://www.java.net/sites/default/files/image_10.png
[14] http://www.java.net/sites/default/files/image_11.png
[15] http://www.java.net/sites/default/files/image_12.png
[16] http://www.java.net/sites/default/files/image_13.png
[17] http://www.java.net/sites/default/files/image_14.png
[18] http://www.java.net/sites/default/files/image_15.png
[19] http://www.java.net/related-topics/open-source
[20] http://www.java.net/topic/programming
[21] http://www.java.net/topic/research
[22] http://www.java.net/blog-community/java-user-groups
[23] http://www.java.net/blog-community/java-desktop
[24] http://www.java.net/topic/community
[25] http://www.java.net/blog-community/global-education-and-learning
[26] http://www.java.net/topic/j2se
[27] http://www.java.net/feed-category/blogs