LAB # 11

General Guidelines for Modeling Class diagrams

Because class diagrams are used for a variety of purposes – from understanding requirements to describing your detailed design – you will need to apply a different style in each circumstance. This section describes style guidelines pertaining to different types of class diagrams.

Figure 1. Analysis and design versions of a class.

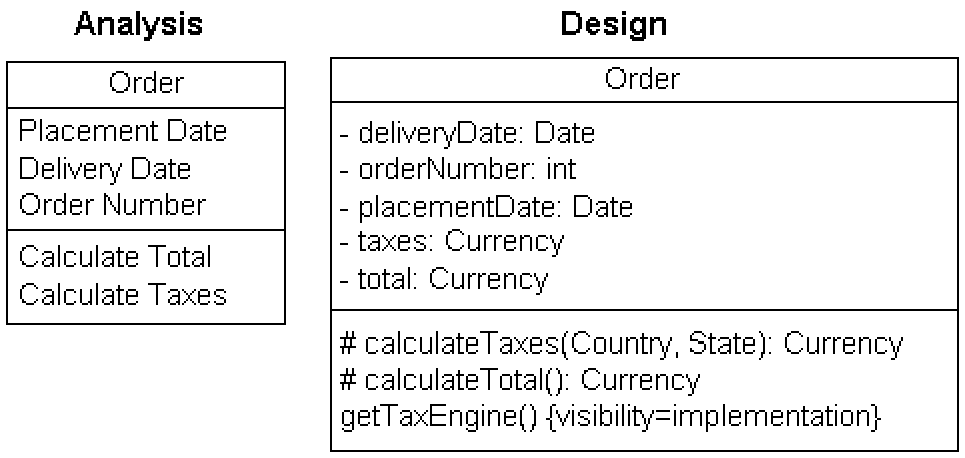
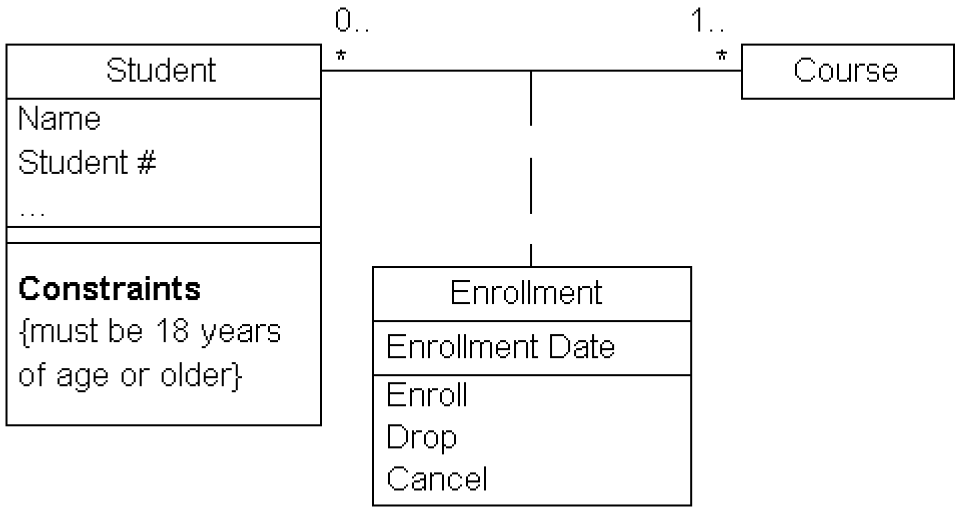


Figure 2. Modeling association classes.



1. Identify Responsibilities on Domain Class Diagrams.
2. Indicate Visibility Only On Design Models.
3. Indicate Language-Dependent Visibility With Property Strings.
4. Indicate Types Only On Design Models.
5. Indicate Types On Analysis Models Only When The Type is an Actual Requirement.
6. Design Class Diagrams Should Reflect Language Naming Conventions. In Figure 1 you see that the design version of the *Order* class uses names that conform to common Java programming conventions such as *placementDate* and *calculateTaxes()*.
7. Model Association Classes On Analysis Diagrams. Figure 2 shows that association classes are depicted as class attached via a dashed line to an association – the association line, the class, and the dashed line are considered one symbol in the UML.
8. Do Not Name Associations That Have Association Classes.
9. Center The Dashed Line of an Association Class.

Class Style Guidelines

A class is effectively a template from which objects are created (instantiated). Although in the real world Doug, Wayne, John, and Bill are all student objects we would model the class *Student* instead. Classes define attributes, information that is pertinent to their instances, and operations, functionality that the objects support. Classes will also realize interfaces (more on this later). Note that you may need to soften some of the naming guidelines to reflect your implementation language or software purchased from a third-party vendor.

Figure 3. The OrderItem class with and without scaffolding code.

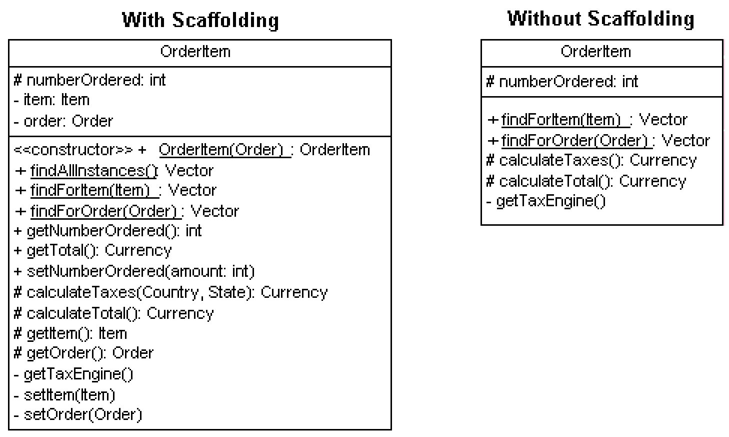

With Scaffolding
OrderItem
# numberOrdered: int
- item: Item
- order: Order
<<constructor>> + OrderItem(Order) : OrderItem
+ findAllInstances(): Vector
+ findForItem(Item) : Vector
+ findForOrder(Order) : Vector
+ getNumberOrdered(): int
+ getTotal(): Currency
+ setNumberOrdered(amount: int)
# calculateTaxes(Country, State): Currency
# calculateTotal(): Currency
# getItem(): Item
# getOrder(): Order
- getTaxEngine()
- setItem(Item)
- setOrder(Order)
Without Scaffolding
OrderItem
# numberOrdered: int
+ findForItem(Item) : Vector
+ findForOrder(Order) : Vector
# calculateTaxes(): Currency
# calculateTotal(): Currency
- getTaxEngine()

Figure 4. Indicating the exceptions thrown by an operation.

1. Use Common Terminology for Names
2. Prefer Complete Singular Nouns for Class Names
3. Name Operations with a Strong Verb
4. Name Attributes With a Domain-Based Noun
5. Do Not Model Scaffolding Code. Scaffolding code refers to the attributes and operations required to implement basic functionality within your classes, such as the code required to implement relationships with other classes. Figure 3 depicts the difference between the *OrderItem* class without scaffolding code and with it.
6. Never Show Classes With Just Two Compartments
7. Label Uncommon Class Compartments
8. Include an Ellipsis (…) At The End of Incomplete Lists
9. List Static Operations/Attributes Before Instance Operations/Attributes
10. List Operations/Attributes in Decreasing Visibility
11. For Parameters That Are Objects, Only List Their Type
12. Develop Consistent Method Signatures
13. Avoid Stereotypes Implied By Language Naming Conventions
14. Indicate Exceptions In An Operation's Property String. Exceptions can be indicated with a UML property string, an example of which is shown in Figure 4.