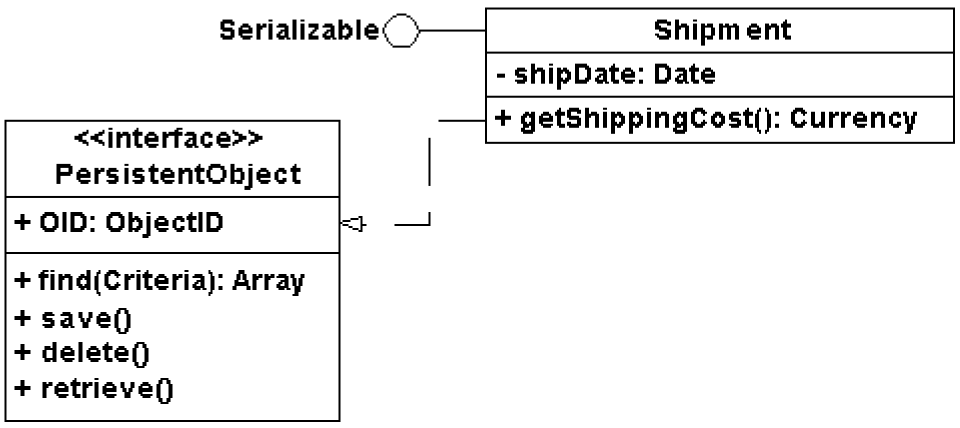
LAB # 12

General Guidelines for Modeling Class diagrams (cont…)

Interfaces

An interface is a collection of operation signature and/or attribute definitions that ideally defines a cohesive set of behaviors. Interfaces are implemented, "realized" in UML parlance, by classes and components – to realize an interface a class or component must implement the operations and attributes defined by the interface. Any given class or component may implement zero or more interfaces and one or more classes or components can implement the same interface.

Figure 5. Interfaces on UML class diagrams.



1. Interface Definitions Must Reflect Implementation Language Constraints. In Figure 5 you see that a standard class box has been used to define the interface *PersistentObject* (note the use of the <<*interface*>> stereotype).
2. Name Interfaces According To Language Naming Conventions
3. Apply "Lollipop" Notation To Indicate That A Class Realizes an Interface
4. Define Interfaces Separately From Your Classes
5. Do Not Model the Operations and Attributes of an Interface in Your Classes. In Figure 5 you'll notice that the *Shipment* class does not include the attributes or operations defined by the two interfaces that it realizes.
6. Consider an Interface to Be a Contract

Relationship Guidelines

For ease of discussion the term relationships shall include all UML concepts such as associations, aggregation, composition, dependencies, inheritance, and realizations – in other words, if it's a line on a UML class diagram we'll consider it a relationship.

Figure 6. Shipping an order.

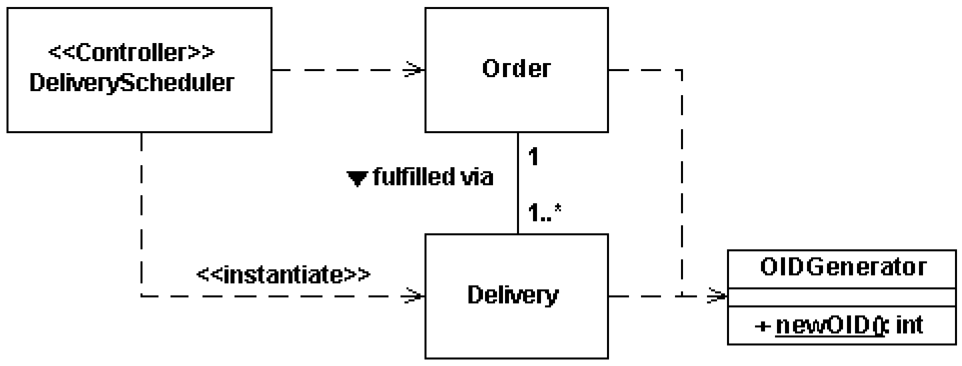
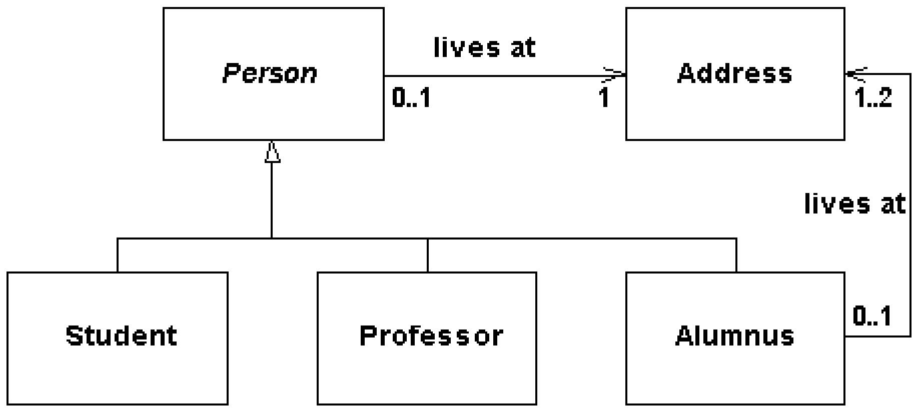


Figure 9. Modeling people at a university.



1. Model Relationships Horizontally
2. Collaboration Indicates Need for a Relationship
3. Model a Dependency When The Relationship is Transitory
4. Depict Similar Relationships Involving A Common Class As A Tree. In Figure 6 you see that both *Delivery* and *Order* have a dependency on *OIDGenerator*. Note how the two dependencies are drawn in combination in "tree configuration", instead of as two separate lines, to reduce clutter in the diagram.
5. Always Indicate the Multiplicity
6. Avoid a Multiplicity of "*"

7. Replace Relationships By Indicating Attribute Types. In Figure 7 you see that Customer has a *shippingAddress* attribute of type *Address* – part of the scaffolding code to maintain the association between customer objects and address objects.
8. Do Not Model Implied Relationships
9. Do Not Model Every Single Dependency
10. Center Names on Associations
11. Write Concise Association Names In Active Voice
12. Indicate Directionality To Clarify An Association Name
13. Name Unidirectional Associations In The Same Direction
14. Word Association Names Left-To-Right
15. Indicate Role Names When Multiple Associations Between Two Classes Exist
16. Indicate Role Names on Recursive Associations
17. Make Associations Bi-Directional Only When Collaboration Occurs In Both Directions. The *lives at* association of Figure 9 is uni-directional.
18. Redraw Inherited Associations Only When Something Changes
19. Question Multiplicities Involving Minimums And Maximums

Inheritance Guidelines

Inheritance models "is a" and "is like" relationships, enabling you to easily reuse existing data and code. When "A" inherits from "B" we say that "A" is the subclass of "B" and that "B" is the superclass of "A." Furthermore, we say that we have "pure inheritance" when "A" inherits all of the attributes and methods of "B." The UML modeling notation for inheritance is a line with a closed arrowhead pointing from the subclass to the superclass.

1. Apply the Sentence Rule For Inheritance
2. Place Subclasses Below Superclasses
3. Beware of Data-Based Inheritance
4. A Subclass Should Inherit Everything

Lab Work and Assignment:

Apply these Set of guidelines to all the problems given in the Exercise