CPS122 Lecture: Class Diagrams in UML

Last revised January 26, 2022

Objectives:

1. To introduce UML Class Diagrams
2. To introduce adornments possible on associations
3. To review the inheritance relationship between classes, and show how to represent it in a class diagram
4. To introduce the reasons why one would want to use inheritance in a design
5. To introduce the realization relationship between a class and an interface
6. To introduce the dependency relationship between classes

Materials:

1. Handout of class diagram for ATM Example
2. Handout of class/object diagram symbols
3. Projectables
4. Navigability and Multiplicity Activity
5. Handout for major league baseball problem and skeleton for diagram
6. Projectable solutions to MLB problem and corresponding code (use .pdf. but do not do full screen)
7. Handout of college class diagram problem
8. Exercises dealing with choice of collections to correspond to UML

I. Introduction

A. In this series of lectures, we talk about how to construct a UML class diagram.

1. We will consider how to represent various properties of associations that we discussed in a previous lecture.

2. We will delve further into the concepts of inheritance (generalization) and interface representation.

3. We will introduce a new kind of relationship between classes: dependency.

4. The point of constructing a class diagram is that it forces us to think about certain key issues, and then to represent our thinking in a pictorial way that guides further design.

5. I will demonstrate using astah to create class diagrams - but you can use another tool (or hand drawing) if you prefer.

B. A preliminary note. UML actually has two similar kinds of diagrams: class diagrams, in which the boxes stand for classes, and object diagrams, in which the boxes stand for individual objects.

1. We will see some examples of object diagrams later in the course.

2. For now, in the diagrams we will be working with, the boxes stand for classes.

3. There are a couple of differences in the way the diagrams are drawn that serve to make this distinction clear. We will deal with these differences shortly.

C. One other thing to be aware of: UML class diagrams using one of two formats:

1. A one compartment box, containing just the class name.

2. A three compartment box, containing respectively the class name, variables that hold its state (instance variables) and its behaviors (methods).

3. Because the one compartment box notation is simpler, we will primarily use this in this lecture, and this is often used in practice.

   a) When the one compartment box is used, the focus is on the relationships between the classes..

b) But there are times when the additional detail of a three compartment box is desirable - e,g, in the case of association classes or when doing detailed design.

D. First, we will look at the quick check questions from the book

1. Quick check question a

   I prefer to use a slightly different way to categorize these

   a) Boundary classes whose objects serve as means by which actors interact with the system - i.e. conceptually they sit on the boundary drawn during use case analysis.

      (1)These may include one or more GUI components

      (2)These may include interfaces to other systems via a network

   b) Controller classes whose objects are responsible for controlling the operation of the system. Typically each use case will be assigned to a controller object - though one controller may be responsible for multiple use cases.

   c) For today, though, we will focus on the classes that are typically discovered early in the process.

      The classes we are focusing on now are often called entity classes because they represent concrete or abstract things.

2. The book suggests two general approaches to discovering the classes that initially belong in the class diagram.

   a) One can consider what objects are involved in realizing a given use case.

      Quick check question b

      When all the objects appearing in each collaboration are combined, the result will be an overall class diagram for the

system. (Note: there will typically be objects that appear in more than one collaboration)

b) One can seek to develop a model of the general domain

c) Either approach should result in the same overall model

E. The book suggests several broad categories of objects to look for in initially developing a class diagram. Specifically, these are useful for recognizing entity objects.

Quick-Check question d [we'll do c in a moment]

1. People
2. Organizations
3. Physical things
4. Conceptual things

Examples from Wheels

ASK

Customers
Bicycles
The hiring of a bicycle

F. However, not everything identified as a possible object should actually be considered as such when developing a class diagram. The book gives a host of reasons for rejecting potential objects.

Quick check question e

G. Ultimately, the class diagram will contain quite a bit of information

1. The classes themselves

2. The attributes of each class

3. The operations of each class

4. Relationships between classes

H. The book suggests an overall process for developing a class diagram

   Quick check question c

   The chapter in the book focused on the first four of these. For now, we will limit our focus to just two: identifying classes, and the relationships between classes.

I. Complete quick check questions before moving on: do questions f-h

II. Relationships Between Classes

A. When we talked about associations, we noted that there are two different sorts of relationship between classes, that we handle similarly but need to keep distinct in our thinking.

   1. There are relationships between *individual objects*. Such a relationship describes how a particular object of one class relates to a particular object of another class. In OO, these are called associations, and we have spent quite a bit of time talking about them.

      Where things get a bit confusing is that when we identify an individual relationship between objects, we are also identifying a relationship between the corresponding classes. The fact that an object of class Book is related to one or more objects of class Author implies that there is a relationship between the *classes* Book and Author such that a member of the one class can participate in this relationship with a member of the other class.

2. There are relationships *between classes*. Such a relationship describes how one whole class of objects is related to another class.

   a) Among humans, the fact that all CS majors are also students is such a relationship.

   b) In the OO world, generalization, or inheritance, is such a relationship.

   c) In the case of a class relationship, all the objects that belong to a given class participate in the relationship in the same way.

3. In drawing a class diagram, we can depict *all* kinds of relationships - even those that are actually relationships between individual objects. (Indeed, the class diagram is the more frequently used type of diagram in UML in general.).

B. In this series of lectures, we will discuss four kinds of relationships (three of which are exemplified in following diagram for the ATM system).

HANDOUT ATM Class diagram

1. *Association (and its stronger form aggregation/composition)* - a relationship between objects.

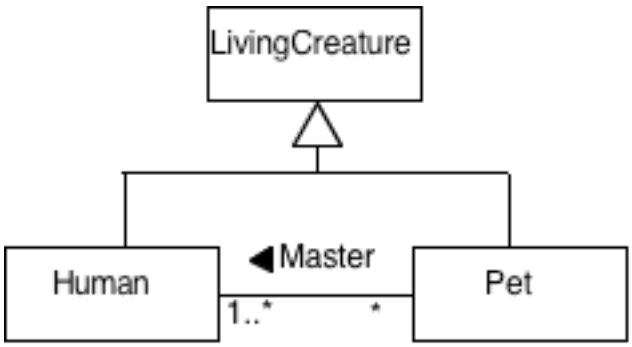
   EXAMPLES FROM CLASS DIAGRAM

   In a class diagram, this kind of relationship is represented by a solid line, possibly with a plain arrow head on one end. There can be multiplicities at both ends.

2. *Generalization* (inheritance) - a relationship between classes. In a UML diagram, this is represented by a solid line with a triangle on one end.

   EXAMPLES FROM DIAGRAM

   Association and genralization are the most common. Something of the difference can be illustrated by the following simple class diagram:



PROJECT

a) The relationship between Humans and Pets is an association - each Human object is potentially associated with some number of Pet objects (possibly zero), and each Pet object is associated with one or more human masters.

b) The relationship between Humans (or Pets) and Livimg creature is that every human or pet object is also a LivingCreature object.

c) The distinction between the two kinds of relationships is very important.

(1)Association is a relationship between objects, typically represented by phrases like "has a" or (in the case of aggregation or composition) "is a part of".

Thus, we say a particular Human has a particular Pet, or a particular Pet has a Human master

(2)Inheritance is a relationship between classes, typically represented by phrases like "is a".

Thus, we say any Human or Pet is a Living Creature

d) Of course, we depict inheritance in class diagrams. But we also depict associations, because the capability of entering into a given

type of association is a property of the class - e.g. an object of the class Pet can enter into a Master association with an object of the class Human, but an object of the class Chair cannot enter into such an association!

3. *Dependency* - a relationship between classes. In a UML diagram, this is represented by a dashed line with an arrowhead on one end.

   EXAMPLES FROM DIAGRAM

4. *Realization* - a relationship between a class and an interface. In a UML diagram, this is represented by a dashed line with a triangle on one end. (Note that the symbol is similar to that for generalization, because realization is similar to inheritance.)

   NO EXAMPLES IN CLASS DIAGRAM - WILL DISCUSS BELOW

C. Everything we will discuss in this series of lectures is summarized in a handout.

1. HANDOUT Diagram Symbols

2. Discuss object and class diagram distinctions

III. Decorations on Associations (and Aggregations and Compositions)

A. How do we represent associations in a class diagram?

B. In the simplest case, an association may simply be drawn as line. But often, the line has one or more decorations *or* adornments that provide further information about the association. [Note: for clarity, as we talk about each type of decoration we will omit others that might otherwise belong in the diagram]

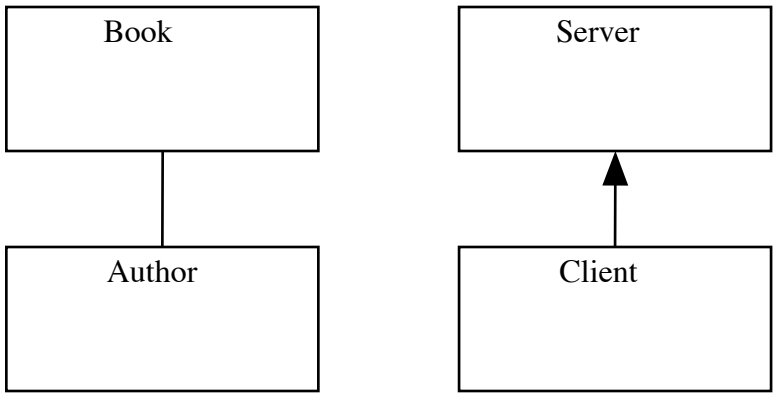
We saw lots of examples of these decorations when we talked about associations, but our focus was on what the decoration means. Now we talk about the actual decoration.

1. Navigability (directionality):

   a) If an association is bidirectional, we just draw a line.

   b) But if it is navigable (directional), we use an arrow to show the direction of navigability.

   c) Examples:



   PROJECT

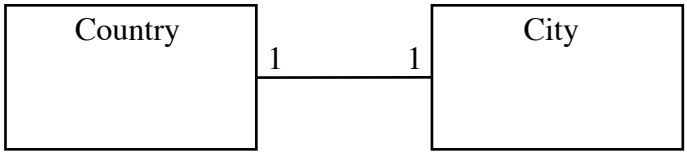
   d) You might ask why we don't draw an arrow on both ends of a bidirectional line. But that would just clutter the diagram - for much the same reason as you don't see one way signs pointing both ways on two-way streets!

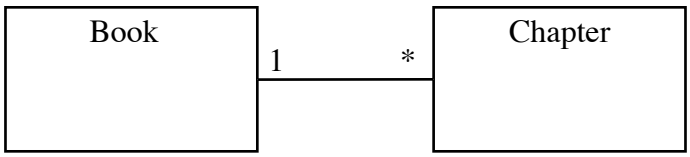
   Thus, the absence of an arrow implies navigability in both directions.

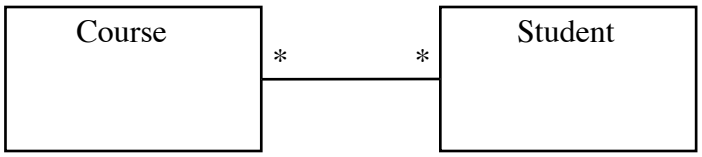
   e) If you are using astah to produce class diagrams, you will notice that it only there are two tools for creating associations - one of which allows you to specify navigability, while the other does not.

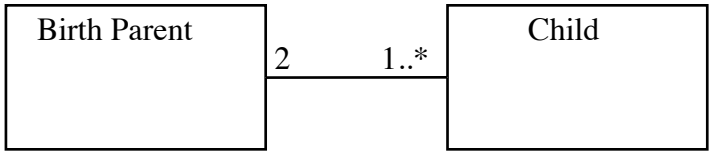
   DEMO with astah

2. Multiplicity: We specify multiplicity by writing numbers on the ends of the association line. We saw many examples of this when we talked about multiplicity under associations.

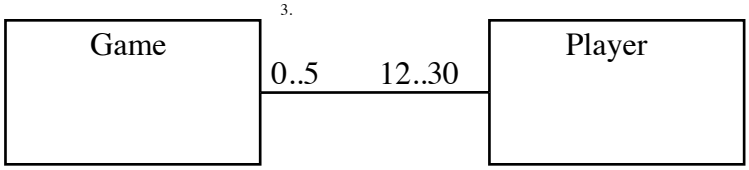








3.



PROJECT ALL

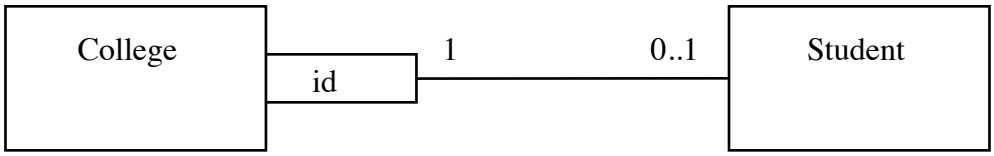
a) The way the multiplicity is read is as follows (using the last diagram as an example: each Player is associated with 0 to 5 games, and each game is associated with 12 to 30 players.

b) Recall that the notation "*" is short for "0..*", and so stands for a relationship that is inherently optional. If the relationship is mandatory, but of unlimited multiplicity, we must use the form "1..*".

c) Also note that some writers use the notation “n” instead of * in a range - so * (= 0..*) is written as “0..n” and 1..* is written as “1..n”.

d) Do Navigability and Multiplicity Activity

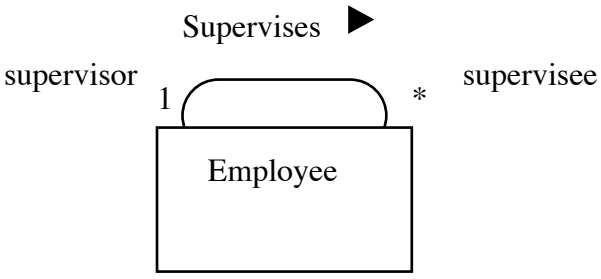
3. Qualified Association: A box on one end contains the qualifier, and the multiplicity on the other end is typically reduced to 0..1 because, for any given value of the qualifier, there is a most one object on the other end.



PROJECT

(Note how the effect of the qualification is to reduce the multiplicity from 1 : n to 1 : 0..1 - for any given id value, there is at most one matching student) Name: Often, the meaning of the association is implicit in the classes that are related, but sometimes an association will be given a name to make its meaning explicit.

4. Role: In the case of a reflexive relationship (and sometimes in other cases) it is necessary to explicitly name the roles at the ends of the association.



PROJECT

Note: Care must be used in drawing a diagram to distinguish between the name of an association and role names. The latter should be drawn near the end of the association line; the former far enough from the ends to be clear that it is not a role.

5. Aggregation/Composition: When we talked about associations, we introduced the notion of aggregation and composition. Aggregation/composition is implemented in the same way as other kinds of association, However, for conceptual reasons in a class diagram we use a notation that distinguishes them.

   a) Review

      (1)What does aggregation mean?

         ASK

         Aggregation is appropriate when we can meaningfully use the phrase “is a part of” to describe the relationship between the part and the whole, or “has a” to describe the relationship between the whole and the part.

      (2)There are two kinds of aggregations - simple aggregation, and a much stronger form called composition.

      (3)Composition is a much stronger form of aggregation. It has the additional characteristic that the “part” has no existence independent of the “whole”. This leads to two additional characteristics:

         (a)Each “part” can belong to only one "whole".

         (b)Each "part" cannot be moved to a different "whole".

(c)The “whole” is responsible for creating and destroying the “parts”.

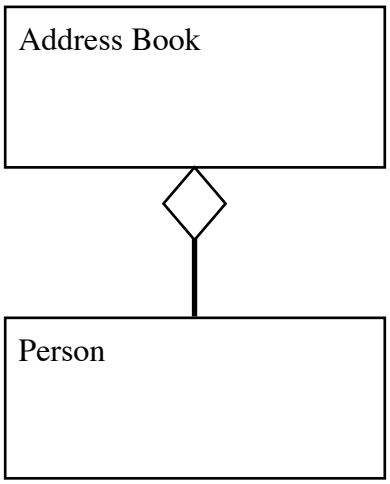
i) The "whole" may potentially create that "part" at any time.

ii) The "whole" may potentially destroy the "part" at any time.

iii)If the “whole” is destroyed, the “parts” are destroyed too

b) In UML, simple aggregation is denoted by a diamond on the "whole" part that is not filled in:

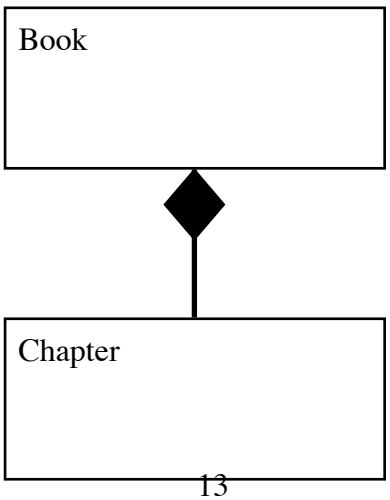
Example:



PROJECT

c) In UML, composition is denoted by using a filled-in diamond on the "whole" part.

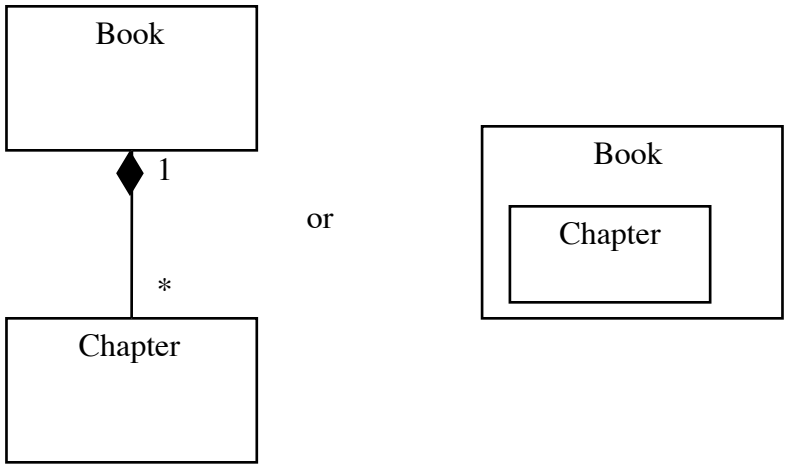
Example:



PROJECT

d) In the case of composition, there is an alternative representation possible in UML. That is to put the box representing the "part" class *inside* the box representing the "whole" class.

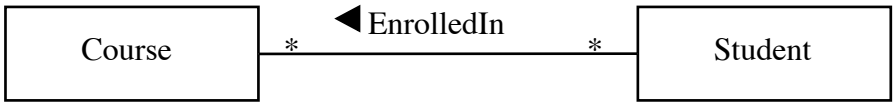
EXAMPLE: Consider the relationship between chapters of a book and the book itself. Clearly, each chapter is a part of one and only one book, and its existence is directly tied to the book of which it is a part. Thus, the association between a book and its chapters is a composition. *Either* of the following UML representations can be used:



PROJECT

The latter representation might be particularly appropriate if the Chapter objects are accessible to the outside world only by *going through* a Book object - i.e. if they don't enter into any relationships with outside objects on their own.

6. Association name: this is written above the middle of the association line, with a filled-in triangle indicating the way the name is to read. For example, the following is read "a student is enrolled in a course.". (The arrow has nothing to do with navigability of the association itself, which is bidirectional in this case.)

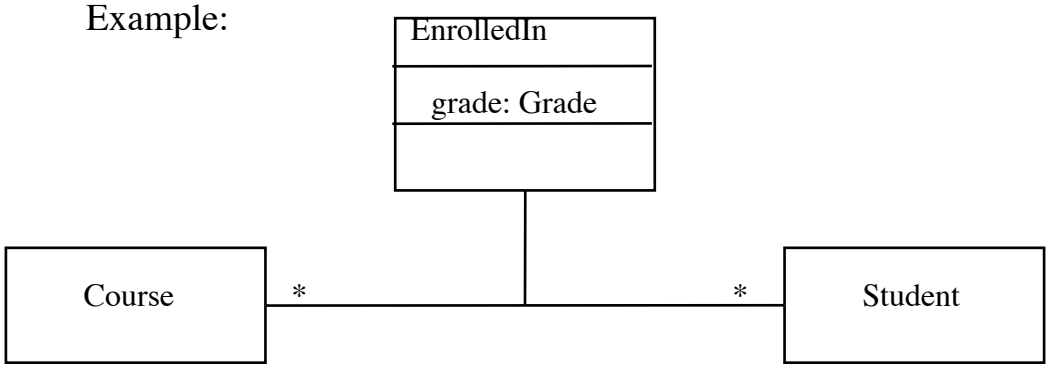


PROJECT

Note that association names typically begin with an upper-case letter, denoting that they are "class like".

7. Association classes: If an association class must be used, then the association class is shown connected to the association line.

Example:



PROJECT
(Note the use of the three compartment box representing the association class, to make it crystal clear that this is a class.)

(Recall the quick check question about a class like Hire)

C. *ON HANDOUT:* Discuss the various associations in the ATM example class diagram.

Note that the relationship between the ATM and its component parts could have been expressed by using the "box within box" representation.

D. Extended Example:

Do Major League Baseball example in groups of 4. At each step, let groups work on then combine results and put on board.

1. Class identification - what classes are needed?

2. Pass out skeleton and fill in association to include needed

   a) Associations themselves (what connects to what)
   b) Multiplicities,
   c) Qualifiers
   d) Aggregation/Composition Symbols
   e) Association Names
   f) Not needed: Navigabilities, Role names, Association classes

3. Then discuss solutions

   a) What associations and decorations are needed

   b) Rationale for choice between aggregation & composition in each case.

4. Discuss resultant code

PROJECT

IV. **Generalization**

A. We saw earlier that there are two different sorts of relationship, that we handle similarly but need to keep distinct in our thinking.

1. There are relationships between *individual objects*. Such a relationship describes how a particular object of one class relates to a particular object of another class.

2. There are relationships *between classes*. Such a relationship describes how one whole class of objects is related to another class.

B. We have been studying associations, which are relationships between objects. We now turn to the study of relationships between classes, of which UML class diagrams recognize three.

C. Probably the most prominent sort of relationship between classes is inheritance, which UML calls "Generalization".

1. Generalization relationships are denoted in UML by using a solid line with a triangle on the base class end.

   NOTE IN HANDOUT

2. Actually, as noted in the book, inheritance can arise in two closely related ways:

   a) Generalization: a base class is created that embodies the common characteristics of a number of similar subclasses. We may discover an opportunity for generalization during design when we notice that two or more classes have a number of characteristics in common, which can be put into a common base class so that they don't have to be duplicated in each class.

   EXAMPLE: Suppose we are developing a system for maintaining course registration information, and create classes "Student" and "Professor". As we develop these classes, we realize they have a lot in common (name, address, phone

number, date of birth, etc.) and so create a generalized class Person that each inherits from.

b) Specialization: a class is created that is similar to its base class, but with certain special characteristics.

We may discover an opportunity for specialization during design when we notice that a class we need to create is very similar to an existing class, with a few variations. Rather than starting from class, we reuse the existing class by inheriting from it and only implementing the things which are different.

EXAMPLE: A Bank might have a special kind of savings account that offers a higher interest rate in exchange for a high minimum balance. If the remaining properties of such an account are the same as those of other savings account, it might be desirable to specialize the class `SavingsAccount` to produce a class `HighBalanceSavingsAccount`.

D. We have already discussed the meaning and mechanics of inheritance in this course. Typically, the decision as to how to use inheritance is made in the design process, and is reflected in the class diagram by the use of the inheritance symbol.

PROJECT

E. Later in this lecture we will talk more about the use of inheritance. When do we use it, and how? But first we want to introduce two more kinds of class relationship.

F. Go over examples in ATM Class Diagram

V. **Realization**

A. The next sort of relationship between classes we want to consider is called *realization* in UML.

1. In many ways, it is similar to inheritance - in fact, in some languages this relationship is represented the same way as ordinary inheritance.

2. Its uses a notation similar to that for generalization, except using a dashed, rather than solid line.

B. In ordinary inheritance, if B inherits from A, then B inherits both A's *interface (specification)* and A's *implementation*. Realization (or what is sometimes called interface inheritance) occurs when we want to specify that a class must provide certain behaviors, without specifying <u>how</u> these behaviors are provided.

There are a couple of clear examples of this we will see later in the Java libraries.

1. The ActionListener interface used with Buttons and MenuItems specifies that an ActionListener object must have a method with signature actionPerformed(ActionEvent), which is called when the Button is clicked or the MenuItem is chosen. However, different ActionListeners may do very different things.

2. In the Collections facility we will consider shortly, List, Map, and Set are interfaces, which can be implemented in a variety of different ways. (In fact, each is implemented in at least two different ways in the Java library, and other implementations could be created by a user.)

C. The standard Java mechanism for realization is to have a class declare that it *implements* an *interface*. (Thus, both the realizing class and the interface it realizes are declared in a special way.)

1. Java actually provides another mechanisms that can be used for specifying an inheritable interface: an abstract class. However, when the realization relationship is intended, implementing an interface is the appropriate facility to use.

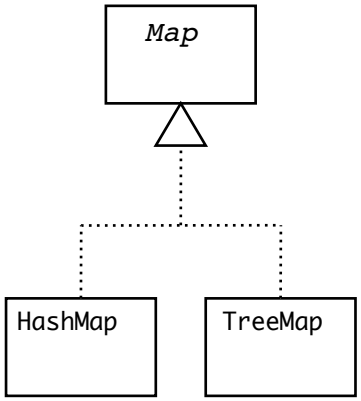
2. Sometimes, in Java, we will use the "implementing an interface" mechanism for inheritance as well as realization. This may be needed because Java does not support multiple inheritance. If we need multiple inheritance to model a particular reality, and one of the classes being inherited is there just for behavior, then implementing it as an interface may let us do what we need to do.

   NOTE: In this case, the UML relationship we are modeling is actually generalization, not realization.

D. In UML, Realization is denoted the same way as generalization, except that the line is dashed, rather than solid.

PROJECT

E. There actually are no examples of this in the ATM Class Diagram. But here is an example from Lab 6:



PROJECT

VI. **Dependency**

A. The final kind of relationship between classes we will consider is *dependency*.

1. Dependency is denoted in UML by a dashed line with an arrow head from the dependent class to the class it depends upon.

PROJECT

2. We say that class A depends on class B if a change to class B's interface could necessitate a change to A. (I.e. A's implementation depends on facilities made available by B.)

B. Dependencies are of various kinds. We will consider only one: *usage dependencies* - where the dependent class *uses* the class it depends upon as part of its implementation.

C. A usage dependency relationship arises when one or more of the following holds:

1. The dependent class has a method that takes an object of the class it depends on as a parameter, and uses that object in some way in implementing the method.

2. The dependent class has a method that returns as its value an object of the class it depends on.

3. The dependent class creates an object of the class it depends on, but only uses it within one method (doesn't keep a reference to it as an instance variable - if it did, we would have an association.)

4. In Java, usage dependencies typically show up in the signatures of methods - as the type of a parameter or a return value - but the object in question is not stored as an instance variable.

D. We take note of dependencies in a UML diagram because they serve to alert us to the fact that whenever we change a class, we need to make sure that we don't need to also change classes that depend upon it.

1. In particular, any time we use an object of a class A as a parameter or a return value of a method of class B, we generally create a dependency from B to A which we should take note of. (No dependency is created if the value is just "passed through" to some other class.)

2. Of course, any time we have an association between objects, we have a dependency between their classes - but we don't take separate note of this - association implies dependency.

3. Likewise, any time we have a generalization or realization relationship, we also have an implicit dependency, which again does not need to be noted separately.

4. We only take note of a dependency when it is present and the classes seem otherwise unrelated to each other.

E. *GO OVER EXAMPLES ON CLASS DIAGRAM HANDOUT*

VII. **Do college class diagram exercise in teams of 4**

VIII. **Do Exercises related to choice of collection types to map UML**