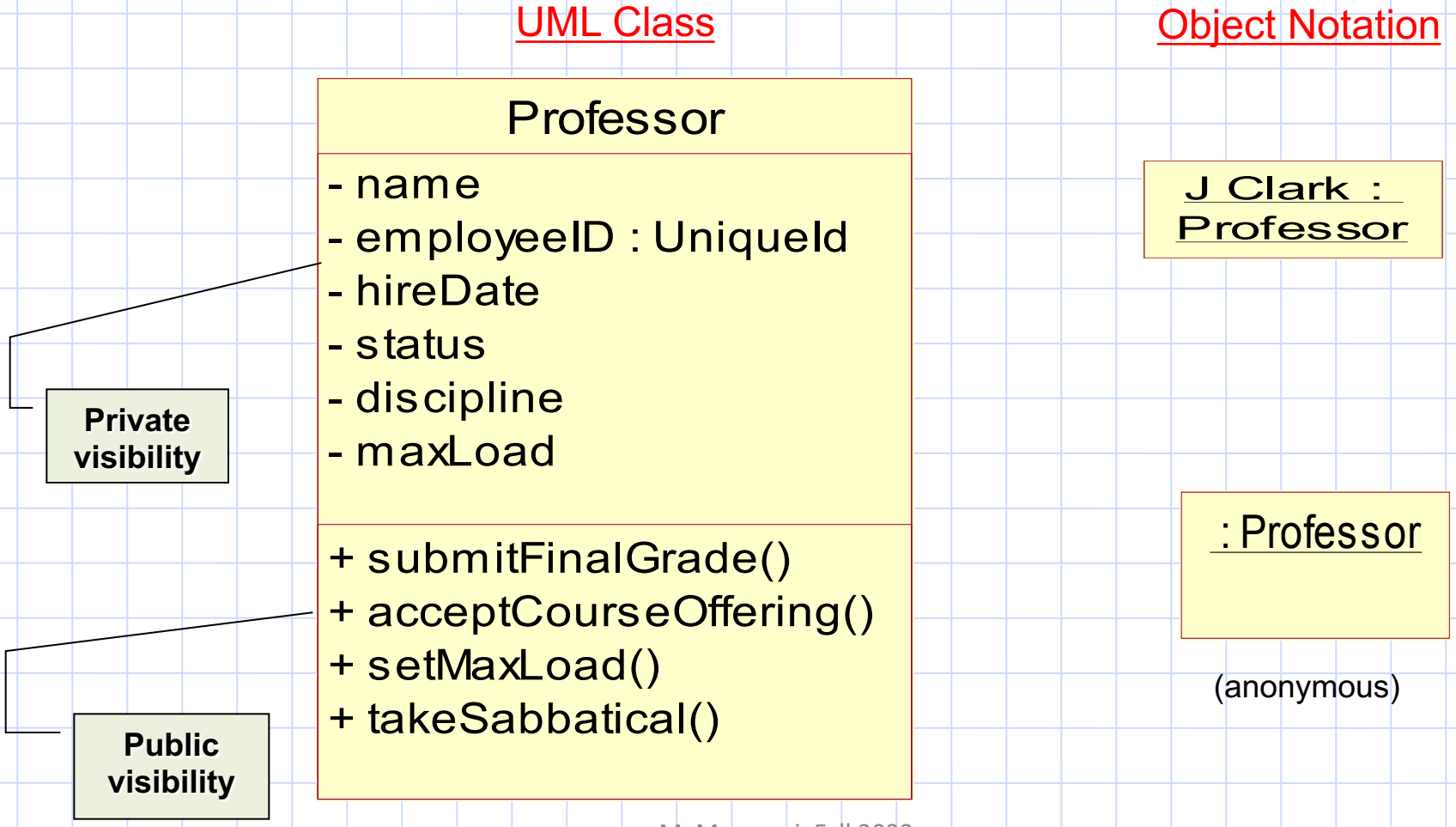
An Overview of
UML Notation for Classes and Objects

- Before we move to the next major topic/chapter of the course, which is"Design Patterns", we need to take a quick look at the proper syntax and usage of UML notations.
- This set of slides only focuses on some of the elements of class diagram, which is essential to know before we discuss design patterns. In future slides other types UML diagrams will be introduced

Representing Classes and Object the UML

- A class is represented using a rectangle with compartments.



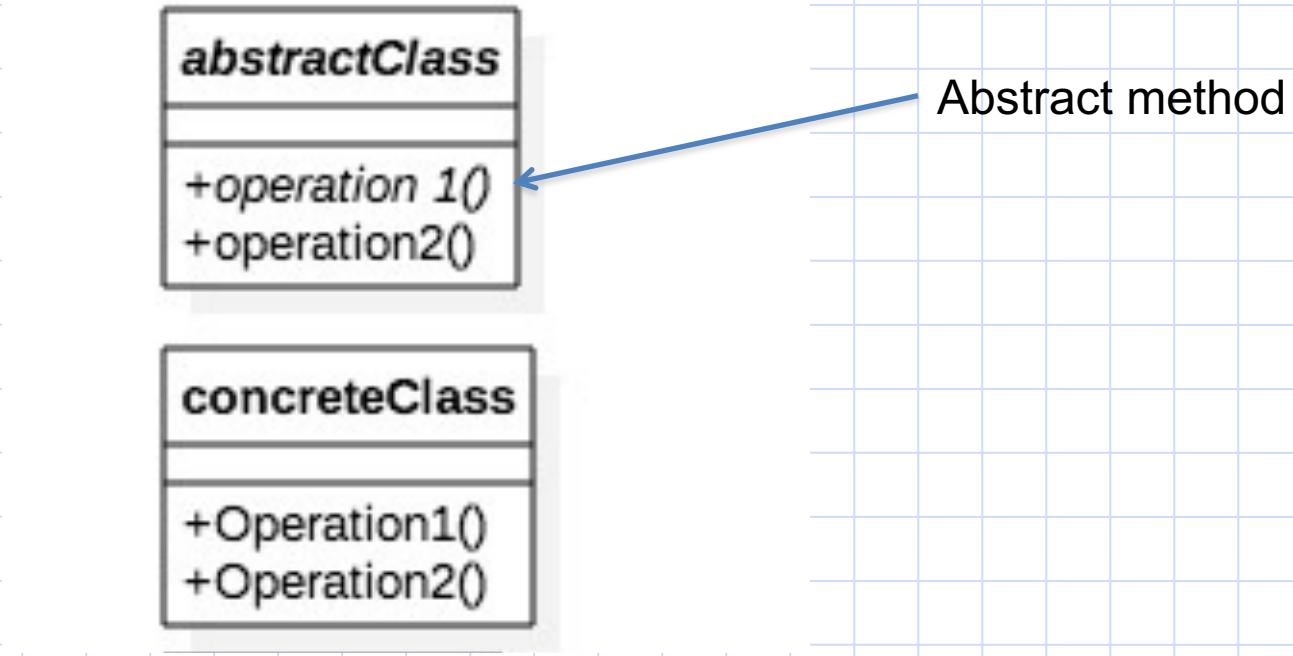
Visibility

- To specify the visibility of a class member (i.e. any attribute or method), these notations must be placed before the member's name:

Public	+
Private	-
Protected	#
Derived	/
Package	~

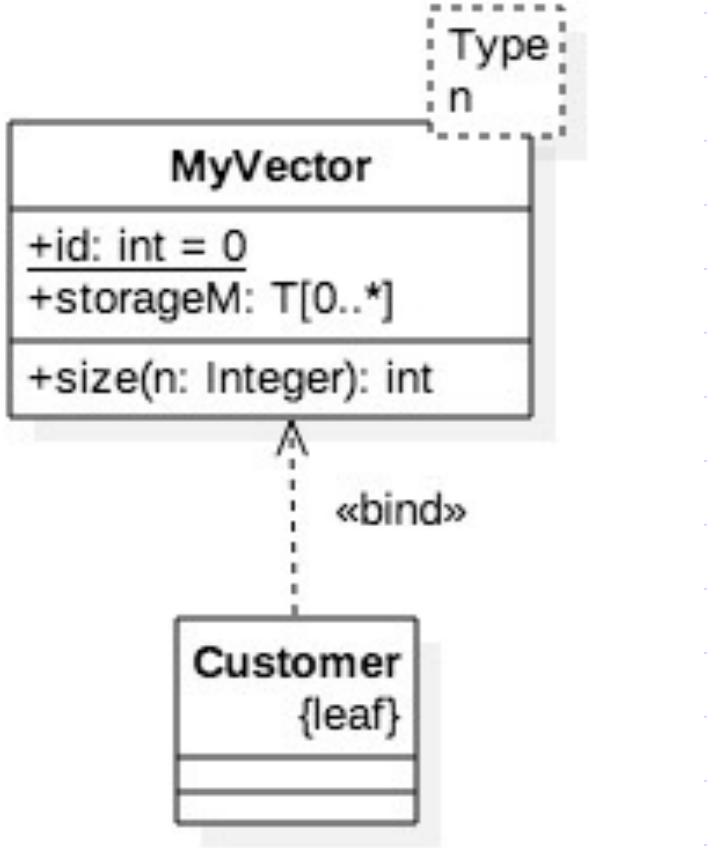
Abstract Classes vs. Concrete Class

- Italic names will be used for abstract classes and methods.
- Using keyword {abstract} is also allowed



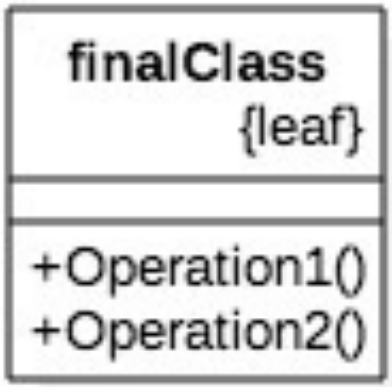
Parameterize Classes

- C++ template classes or Java generic types can be represented by parameterized class notation:
- This concept is most obviously useful for working with collections in a strongly typed language.



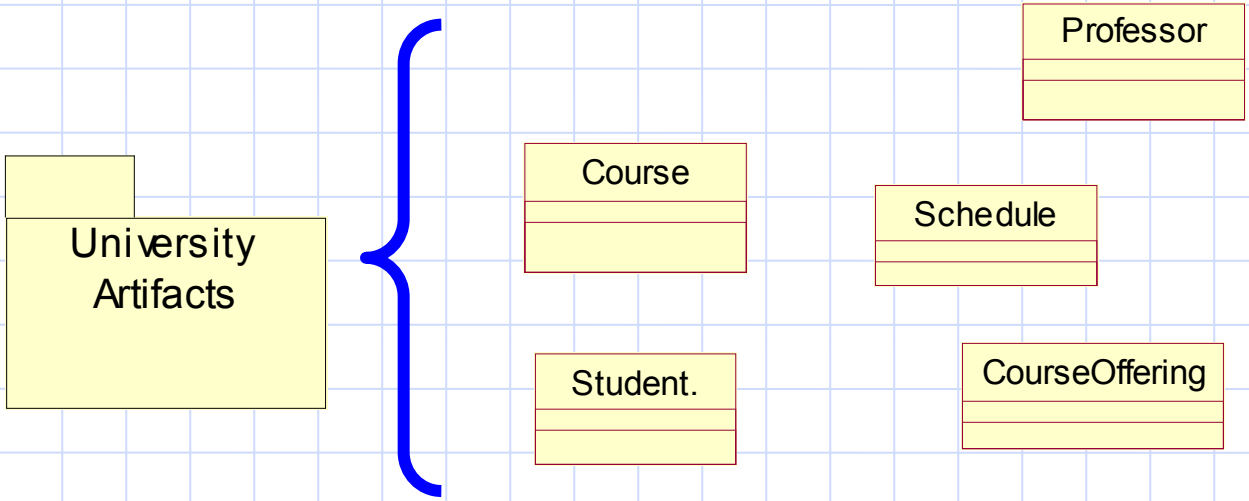
Final Classes

- A class marked final cannot be subclassed:
  `final class ClassName { }`
- Final classes and methods are more secure, since their behavior cannot change.
- Final classes and methods can be optimized by the compiler.



Representing Packages in UML

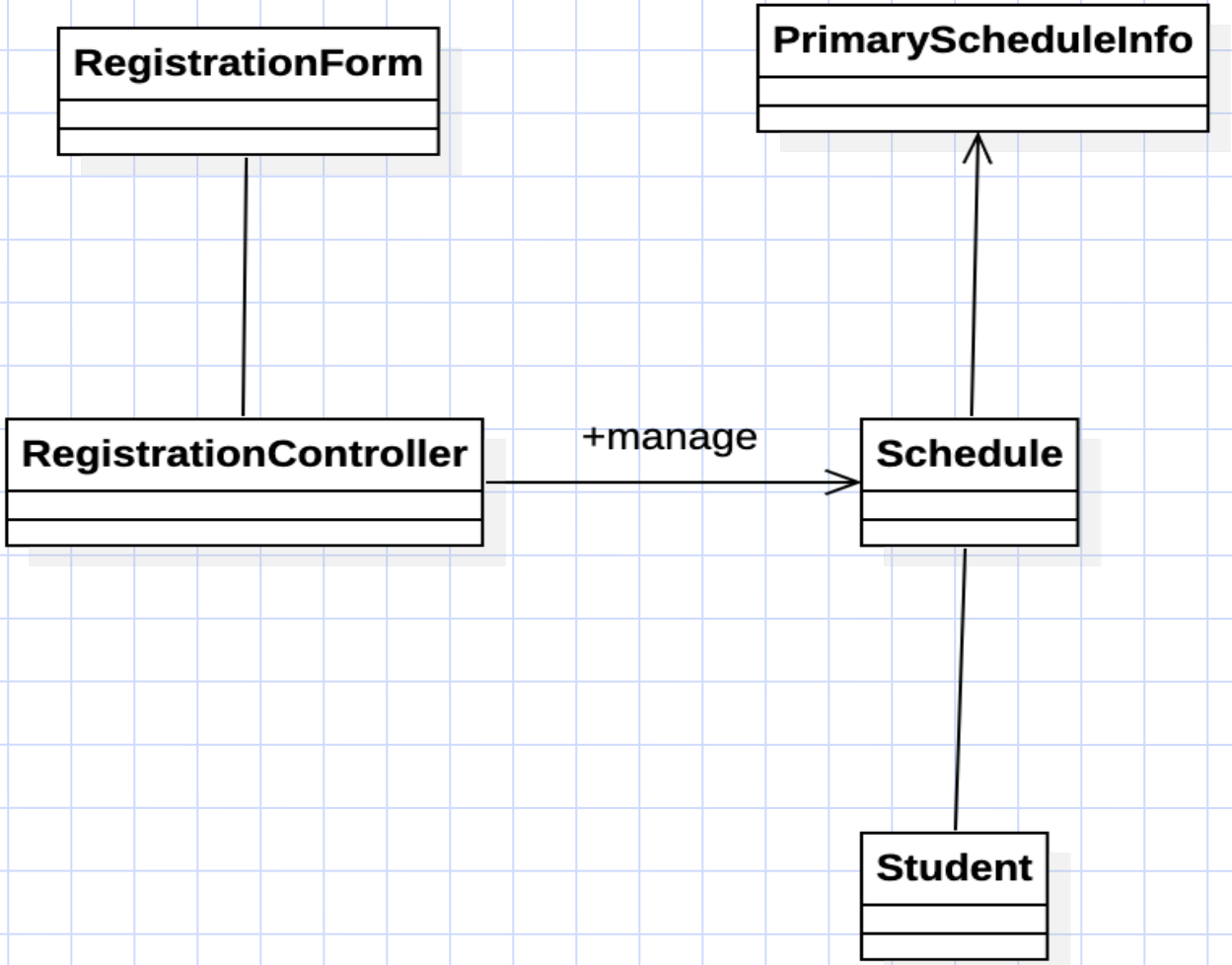
- A package is a general-purpose mechanism for organizing elements into groups.
- It is a model element that can contain other model elements.



A package owns its elements and can even own other packages.
If the package is destroyed, the element is destroyed, too.

What Is Navigability?

- Indicates that it is possible to navigate from an associating class to the target class using the association



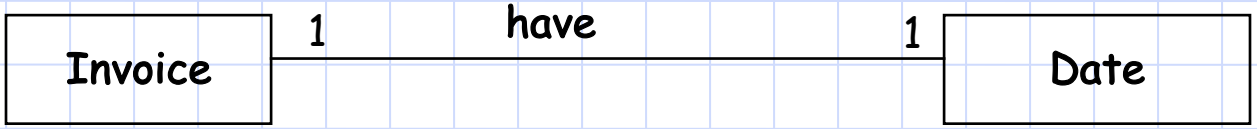
What is Cardinality?

What is Multiplicity?

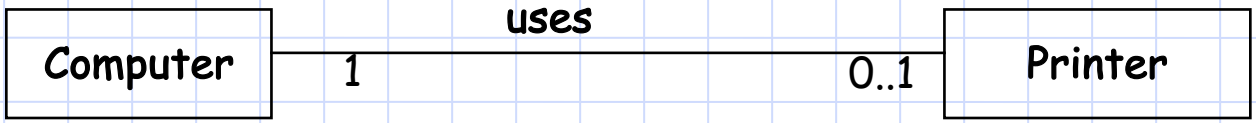
- Multiplicity or cardinality answers two questions.
  - Is the association mandatory or optional?
  - What is the minimum and maximum number of instances that can be linked to one instance?
- The following example expresses each student "MAY" register for many courses:

More Examples of Multiplicity

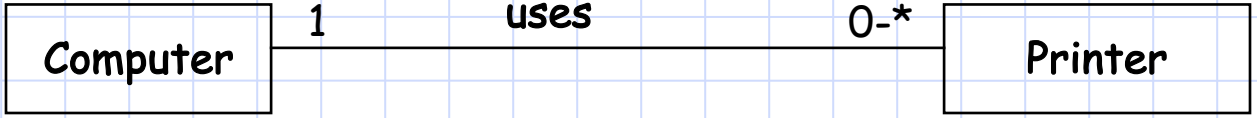
- Example of 1 to 1: Each Invoice must have a Date:



- Example of 1 to 1: Each Computer may be using a printer:



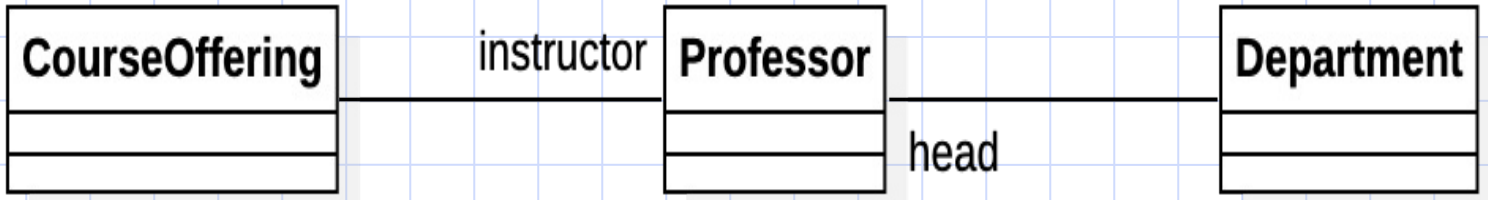
- Example of 1 to many: Each Computer may use several printers:



- Example of many to many o many Student may register for many courses:

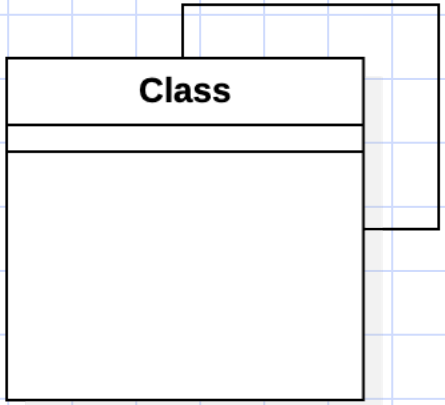
What Is a Role?

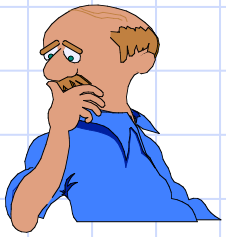
- A role specifies the face that a class plays in an association.
- Role names are typically nouns or noun phrases.
- A role name is placed along the association line close to the class it modifies.
  - One or both ends of an association may have role names.



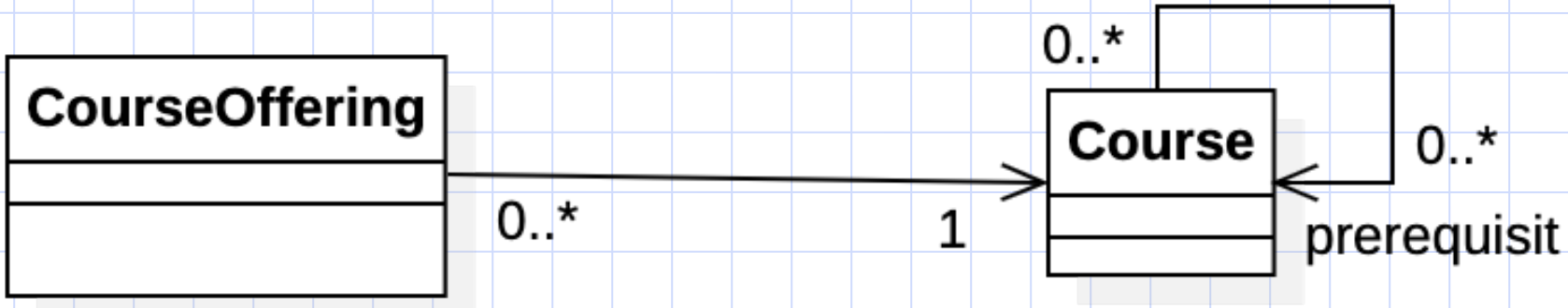
Reflexive Associations and Roles

- In a reflexive association, objects in the same class are related.
  - Reflexive associations indicate that objects in the same class collaborate together in some way.
  - Role names must be used in a reflexive association.





What Does Following Diagram Tell You?

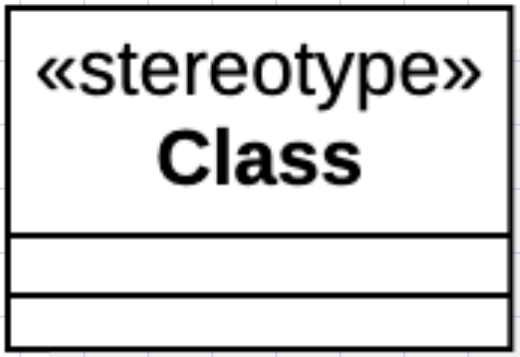


UML Static Notation

- Class (i.e. static) methods and fields are indicated by underlining

What Are Stereotypes?

- Stereotypes define a new model element in terms of another model element.
- Sometimes, you need to introduce new things that speak the language of your domain and look like primitive building blocks.

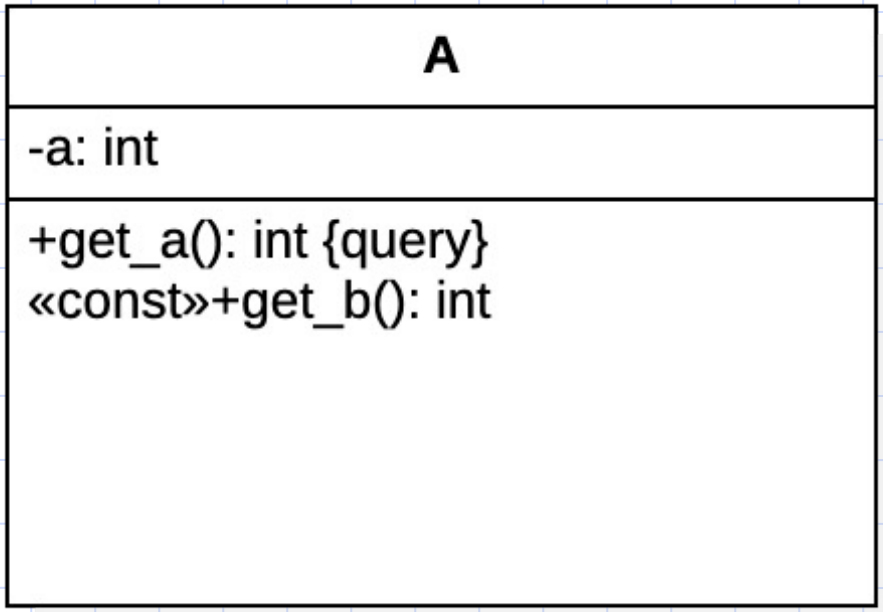


A Few Other Note about UML Notations

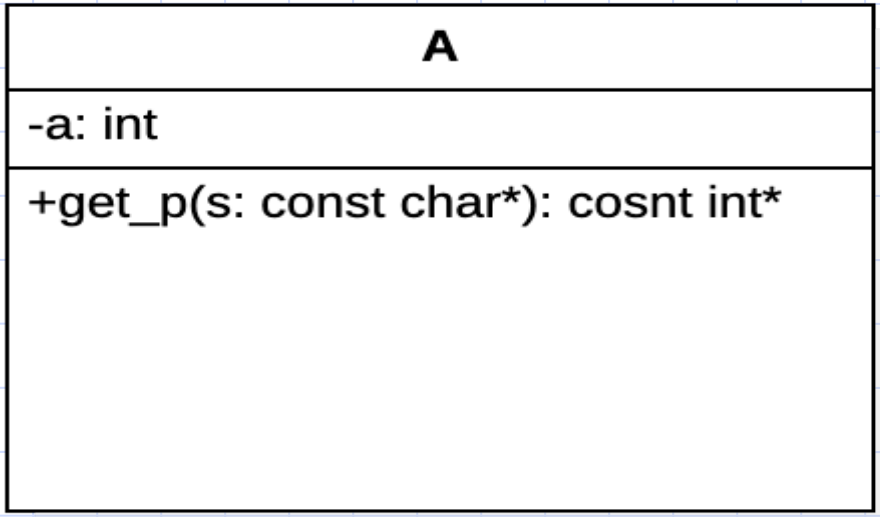
Read only functions

- How should I use a const member function in a class:

```
class A {
  private:
      int a;
      int b;
  public:
      int get_a() const;
      int get_b() const;
};
```



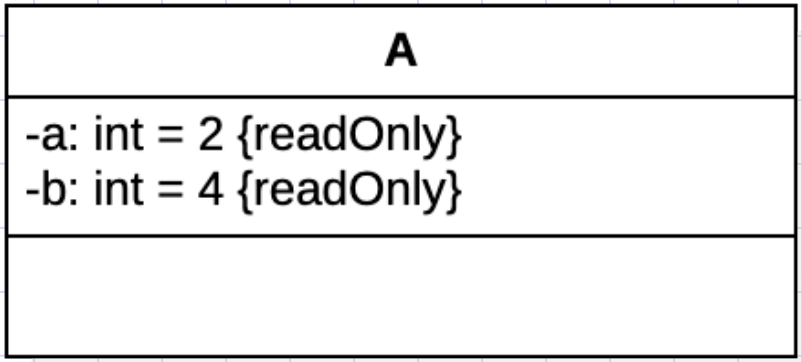
Const Argument and Return Type




```
class A {
  private:
      int a;
      int b;
  public:
      const int* get_p(cosnt int* a);
};
```

Using Const Data Members

```
class A{
private:
      const int a = 2;
      const int b = 4;
public:
      ...
};
```



How to Show Files with Only Constants or Global Functions

- In reverse engineering, if there is a file with only a few constants, or some global function (no class member functions), you can either use an interface or class notation.
- For each constant declaration you can use a final keyword.