

UML Class Diagrams

CPSC 2150

What is UML

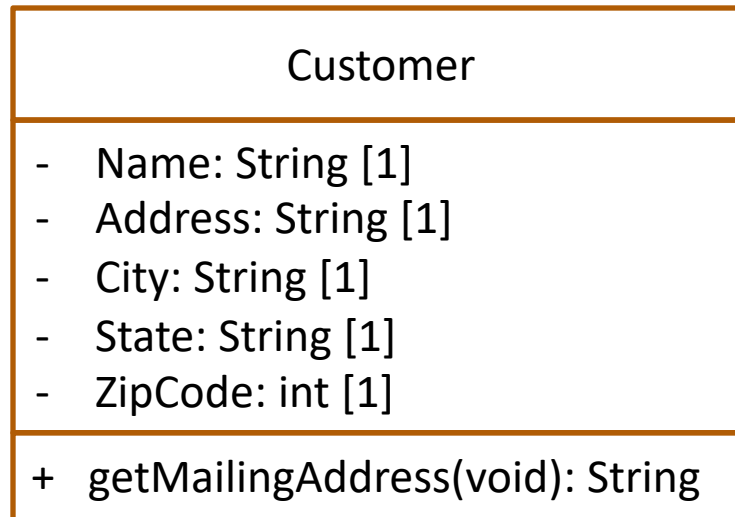
- Unified Modeling Language
- Set of graphical notations
 - All backed by one model
 - Used to design software
 - Used to describe software
 - Particularly useful for Object Oriented Code
- We'll use it throughout this semester
- This is not a UML course, but UML is very useful for the processes we will learn

Class Diagrams

- Class Diagrams are one of the most common UML diagrams
- We use them to show a class we will create in our system
- It's helpful to identify the classes and their attributes and methods before we start writing the code
 - Part of our design process
 - Work out how to solve the problem before you focus on the details of the code
 - Break the problem up into smaller pieces

Class Diagram

- 3 Parts
 - Class Name
 - Attributes
 - Methods



Specifying Attributes

- Have some idea already, need to refine
- Attributes have a name and a type
- Also (optional):
 - Multiplicity
 - Visibility
 - Default value
 - Other property
- Denoted by
- `<visibility><name>:<type>[<multiplicity>] = <default>{<other>}`

Multiplicity of Attributes

- N – exact number
 - 0, 1, 2 ...
- N..M – exact range
 - 0..1, 1..5, etc.
- * - 0 or more
 - * can be used in a range
 - 1..* is 1 or more
- In code, multiplicity is usually handled by collections
 - But in UML, we do not put the collection data type

Visibility

- - indicates private
 - Only available to the class itself
- # indicates protected
 - Available to the class itself and any sub classes that inherit this class
- + indicates public
 - Available to anyone
 - Not common for attributes
- ~ indicates package
 - Available to other objects in the same package

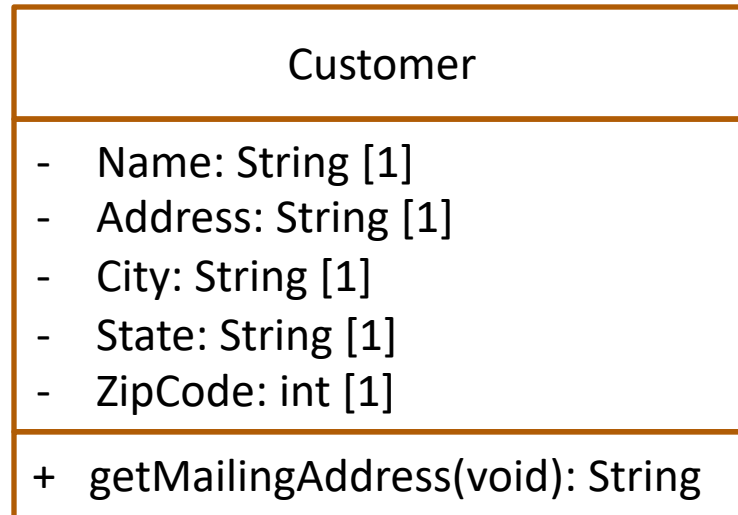
Other property

- Any sort of constraint you can think of
- {readOnly}
- {Ordered}
- {Unordered}
- {Unique}
- {NonUnique}

Signatures

- Signatures correspond to methods/operations of the object
- Signature contains:
 - Visibility
 - Name
 - Parameter types
 - Return type
- Denoted with
- `<visibility><name>(<parameter types>):<return type>`
`{<property>}`

Class Diagram

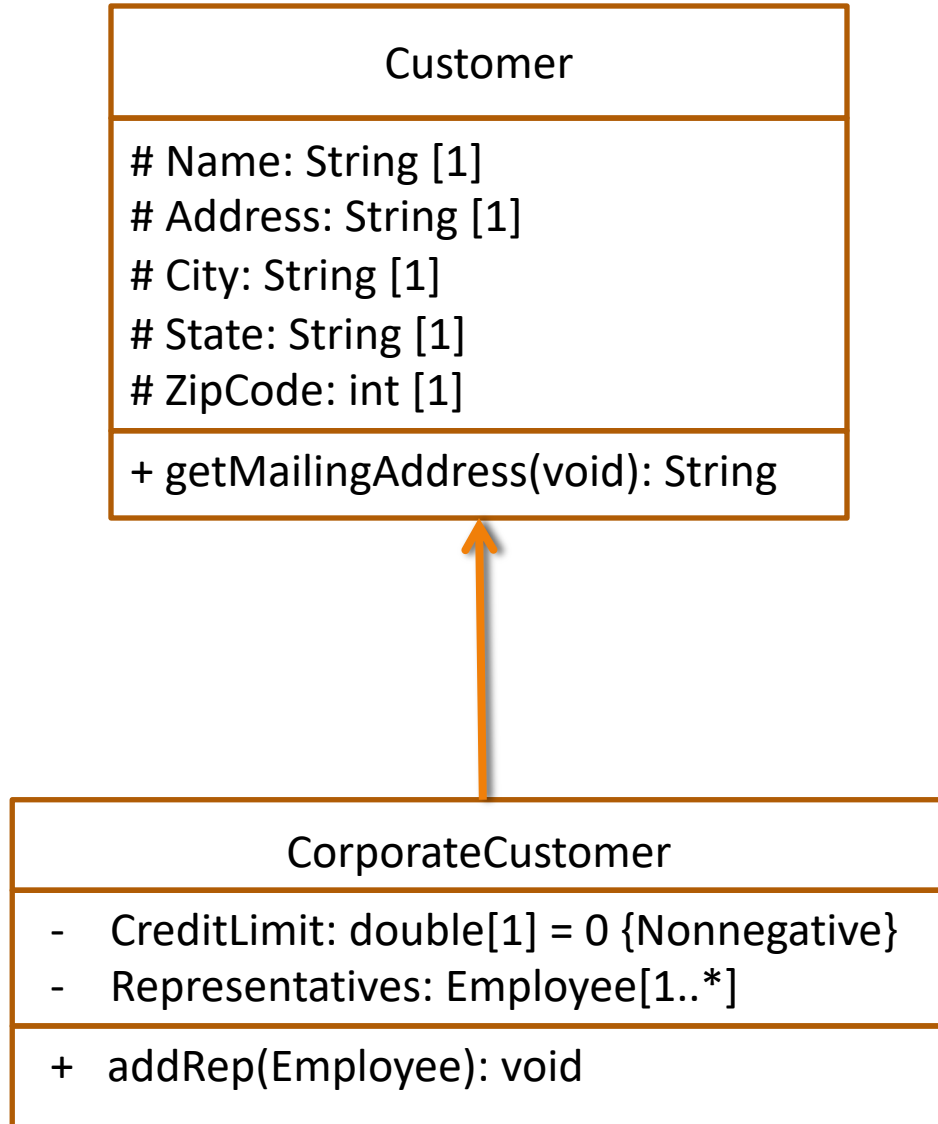


Inheritance

- Inheritance between classes is shown with:



Inheritance

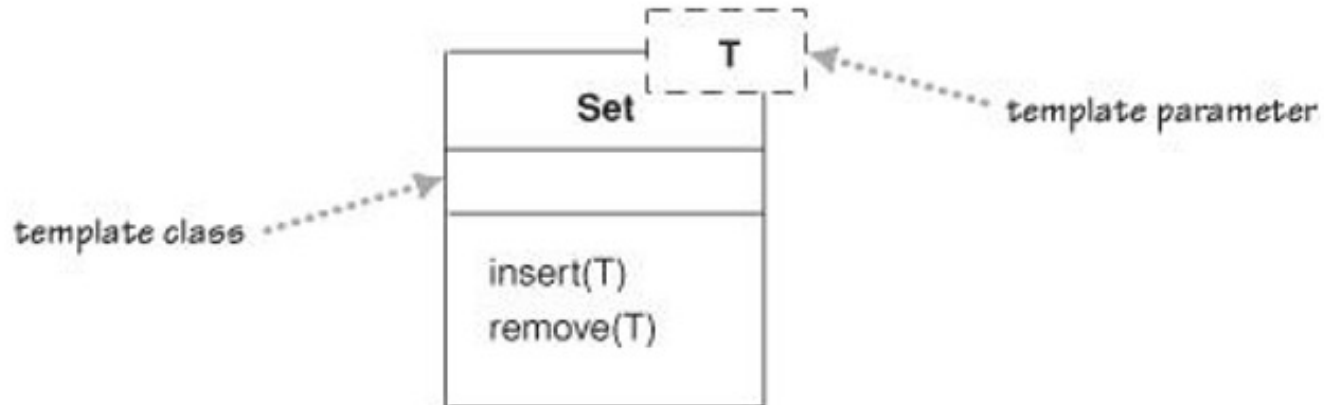


Advanced Notation

- Static Attributes and Operations
 - Attributes or operations that apply to the class, rather than the instance of the class
 - Helpful for utility objects
 - Noted by underlining the operation or attribute
- Derived properties
 - An attribute of the object conceptually, but not stored in memory
 - A property that could easily be calculated
 - For instance, Age is derived since it depends on the birth date and the current date
 - Noted by adding “/” to the beginning of the name

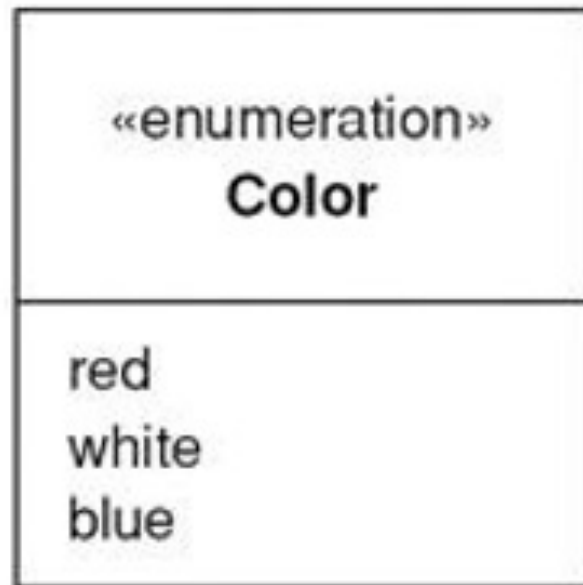
Advanced Notation

- Template/Generic Classes



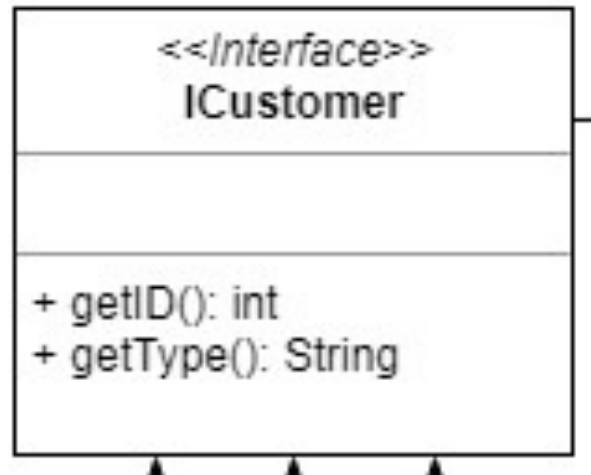
Advanced Notation

- Enumeration
 - A set of fixed values
 - No other properties



Advanced Notation

- Interfaces
- Look the same as a class diagram with a few minor differences
 - <<interface>> above the name of the interface
 - All methods must be public
 - All attributes must be public and static



The End

- In this course you'll be asked to make class diagrams for every assignment
- Work through the design of your system before you write the code
 - Like outlining a paper
 - May seem unnecessary now with smaller simpler programs
 - It's a good skill to practice before it becomes necessary
 - Practice designing small programs, so it won't be as hard with large programs with hundreds of classes