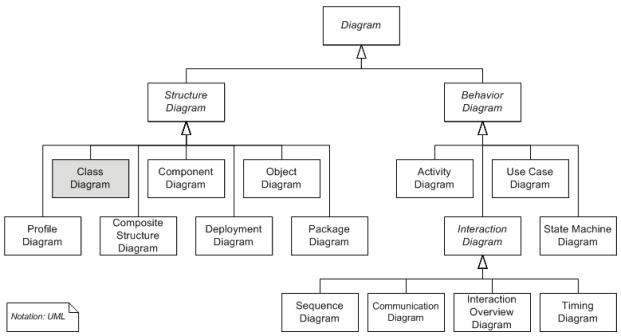
CDE Services
Consulting
Development
Education

Class Diagrams Basics

Miroslav Biňas
(c) 2010 - 2017

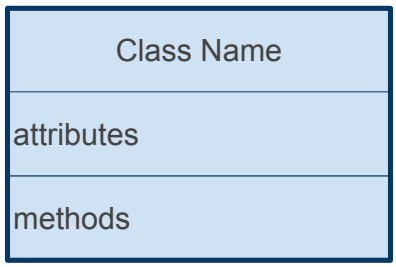


CDE Services

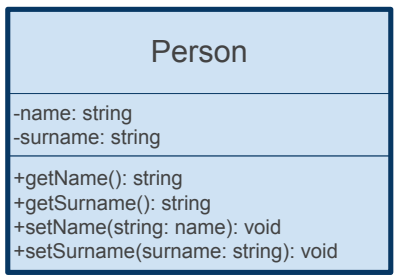
Introduction

- one of the common UML diagrams
- **class diagram** - describes types of the objects in the system and relationships between them
- contains
  - **features** - properties (attributes) and methods
  - **restrictions** in usage

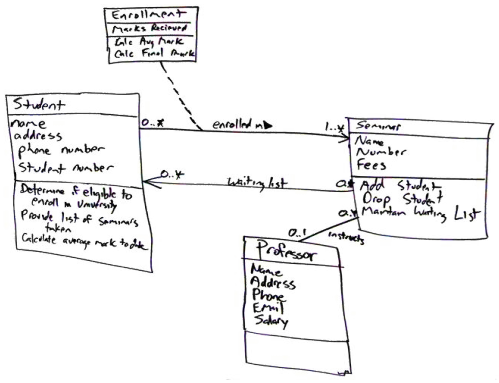
Class Diagram Structure



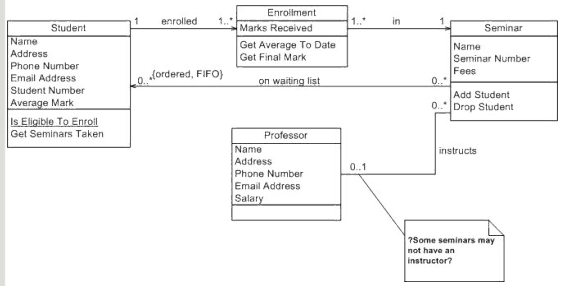
Class Diagram Example



Sketch of Conceptual Class Diagram



Initial Conceptual Class Diagram



Properties

- structural features of class
  - class entries
- two notations:
  - **attributes**
  - **associations**
- in diagram each notation looks different, but meaning is the same

Attributes

- notation using attributes describes properties with line of text inside of the class box
- attribute syntax:

```
visibility name: type
multiplicity = default-value
{flags}
```

- Example:

```
- name: String [1] = "no
name" {readOnly}
```

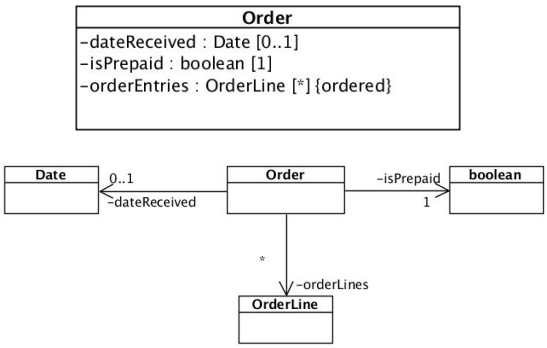
Attribute Notation Explained

- `visibility` - public (+), private (-), protected (#)
- `name` - mandatory
- `type` - type of attribute value/object
- `multiplicity`
- `default-value` - value for newly created object
- `flags` - other features of attribute

Associations

- solid oriented line (with arrow) between two classes oriented from source class to destination class
- destination class specifies type of property
- at the destination class is written:
  - property name
  - multiplicity

Associations Example



Tips for Usage of Properties

- Q: when to use attributes and when associations?
- attributes - mainly for value types or built in reference types
- associations - mainly for special classes (`Customer` and `Order`)
- what to use and what to hide (attributes) is based on what would you like to highlight

Multiplicity

- multiplicity of property defines, how many objects can be stored in property
- common values:
  - `1` - exactly one instance, default
  - `0..1` - no instance or one instance
  - `*` - zero or more instance
- multiplicity defines range of instances - `lower_limit..upper_limit`

Forms of Multiplicity

- **optional** - lower limit starts at `0`
- **mandatory** - lower limit start min. at `1`
- **single-valued** - upper limit is `1`
- **multivalued** - upper limit is greater than `1` (usually `*`)

Multiplicity Flags

- `{ordered}` - set of entries in order
- `{nonunique}` - not unique
- `{unordered}` - default, not ordered
- `{unique}` - default, unique

Properties Implementation

```
public class Order {
  private Date dateReceived;
  private boolean isPrepayed;
  private List<OrderLine>
            orderEntries;
}
```


Bidirectional Associations

- possible to use without arrows
  - useful in conceptual models
- bidirectional association is pair of properties inversely connected
  - `Car` has property `owner:Person [0..1]`
  - `Person` has property `cars:Car [*]`

Named Associations

- possible to use instead of properties
- relation is marked with verb
  - arrow shows orientation
  - it can be used in sentence describing relation
  - ex. `Person` owns `Car`

Operations

- describes class behavior (methods)
- UML syntax:
  `visibility name (params): return-type {flags}`
- example:
  `+ hasCar(car: Car): boolean`
- in conceptual models do not use operations for interface definitions - use several words for responsibility description instead

Operation Notation Explained

- `visibility` - public (+), private (–), protected (#)
- `name` - mandatory
- `params` - operation parameters
- `return-value` - type of return value
- `flags` - other features of operation

Operation Parameters

- similar to attributes
- syntax:
  `direction name: type = default_value`
- `name, type, default_value` - same as attributes
- direction - defines, if parameter is input - `in` (default), output - `out` or `inout`

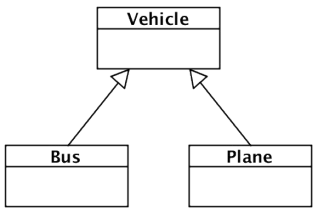
Direction Semantics I.

- `in` (input) parameter is:
  - used as input parameter
  - used during operation
  - not changed during operation
  - default
- `out` (output) parameter is:
  - used as output parameter - as resulting value of operation
  - can be changed during operation

Direction Semantics II.

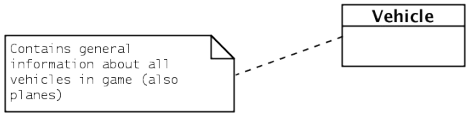
- `inout` parameter is:
  - used as input and output parameter
  - used during operation
  - used as output parameter - as resulting value of operation
  - can be changed during operation

Generalization



- describes **inheritance,** IS-A relationship
- arrow orientation is from child class to parent class

Notes and Comments

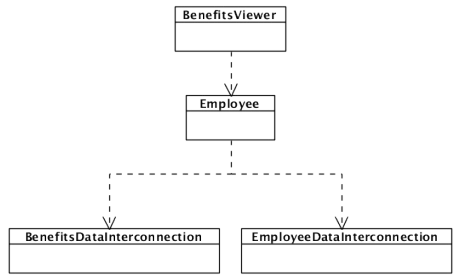


- notes are comments
- connection with element is through dashed line
- possible to use in any UML diagram

Dependency

- exists between two elements, when change in one element (**source**, **provider**) leads to change in second element (**destination**, **client**)
- dependency examples: class sends message to other class, class contains other class as data or parameter in operation/method
- risk - domino effect, cyclic dependency
- goal - keep dependencies at minimum
- basic dependency is not transitive

Dependency Example



Dependency Stereotypes

- «call» - source calls operation on destination
- «create» - source creates instances of destination
- «instantiate» - source is instance of destination
- «permit» - destination permits to source own private methods/attributes
- «substitute» - source can replace destination (inheritance)
- «use» - source requires for own implementation destination

Constraints

- most of stuff in designing class diagrams express **constraints**
- associations, attributes, generalizations helps to refine constraints
- anything can be used to express constraints
  - constraints must be located in {}
  - programming language, *Object Constraint Language* (OCL)
- {title: can't be uppercase}

When to use Class Diagrams

- don't try to use all possible notations
- conceptual class diagrams are very useful in process of analysis
- don't create models for everything - focus on key points

Questions?

References

- Class Diagram (Wikipedia)
- UML 2 Class Diagrams
- UML basics: The class diagram