Contents

Contents

1 Three-tier Architecture

Three-tier Architecture

- Well-designed program (information system) is composed of several logically separated parts:
    - Visualisation of data to user (UI, WEB, thin client, . . .),
    - data storage and maintenance (database, filesystem, . . .),
    - data processing.
- Well-managable program should be divided into independent parts – blocks, tiers.
- Tiers are communicating through the interfaces.

1.1 Presentation Tier

Presentation Tier

- *Presentation tier* displays contents to user, do prints etc.
- It has tools for cooperation between user and application – user can send commands to IS.
- In Server-Client systems is presentation tier in client side.

1.2 Business (Application) Tier

Business (Application) Tier

- *Business Tier* implements main logic of application.
- It takes commands from presentation tier and reacts on them.
- All (or most) computations is held here as well as validation etc.
- It coordinates data flow between presentation and data tier.
- In Server-Client systems is business tier mostly in server side, part is on client side.

1.3 Data Tier

Data Tier

- *Data Tier* is responsible for all data maintenance.
- Data are served to business tier (and then to presentation tier).
- It takes all entered (changed) data from business tier as well as other request for data manipulation.
- In Server-Client systems is data tier in server side.

Three-tier design requirements

- All tiers are interchengable.
  - Data tier that works with data on HDD,
  - data tier that works with Oracle DB,
  - data tier that works with streamer memory,
  - ...
- *Interface must be confirmed while developement.*
  - Best usage is creating abstract class that specifies interface (or make interface in Java-like languages).
  - Inherit class from the abstract one – this should implement all methods.

2 Creating object dynamically

Creating object dynamically

- Enhanced syntax of `New` procedure for using with objects:

  ```
  procedure New(p:^CClass, Init:Constructor);
  ```

  - This allocates object from class CClass on heap and invokes constructor.

- Enhanced syntax of function `New` for using with objects:

  ```
  function New(T:^CClass, Init:Constructor):^CClass;
  ```

  - This allocates object from the class CClass, invokes constructor and returns pointer to it.
  - Used with inheritance. It is possible to create object from descendat class and store it into variable with type of ancestor:

    ```
    varAncestor:=New(PDescendant, Init);
    varAncestor:=New(PDescendant, Init(10));
    ```

2.1 Destructors

Destroying Objects & Destructors

- Object should be freed when no longer needed.

```
procedure Dispose(p:^CClass);
```

- Each object should free all allocated memory prior to own destruction.
  - This could be accomplished with a method
  - . . . or with special ”method” called *destructor*.

```
type CClass=object
       ...
       destructor Done;
       ...
     end;
procedure Dispose(p:^CClass, Done:Destructor);

Dispose(object, Done);
```

- Destructors are (almost) everytime *virtual*.

3 Class Diagram in UML

Class Diagram

Class Diagrams shows class relations:

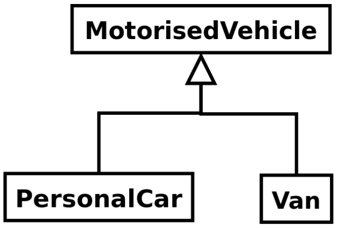
- inheritance structure,
- associations, and
- relations between the whole and parts.

3.1 Generalisation Construction – Inheritance

Generalisation Construction

Generalisation:

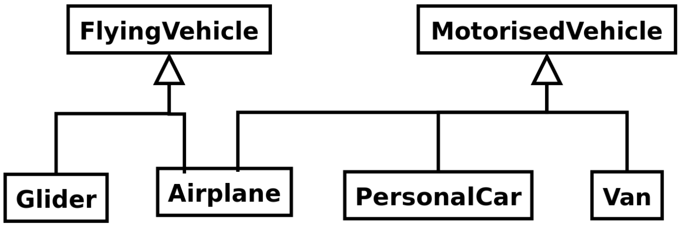
- Shows inheritance.
- Is written as empty triangle arrow.
  - The arrow points to ancestor.



Multiple Inheritance

Multiple Inheritance:

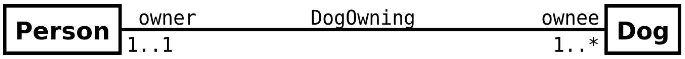
- Descendant inherits everything from its' ancestors.
- This construction is not recommended – it is possible to replace it by aggregation or interfaces.



3.2 Association Construction

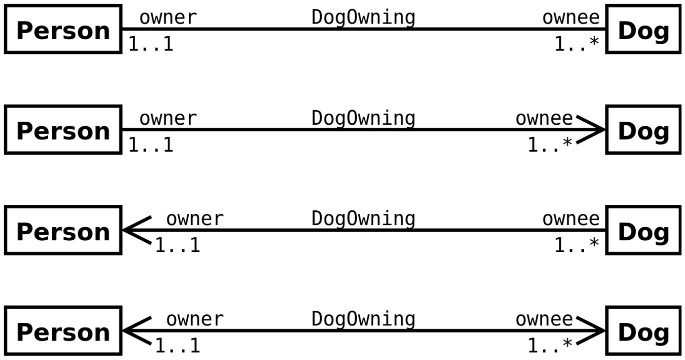
Association Construction

- Association in UML represents mutable population of interconecting relations between object.
- E.g. dogs and its' owners:
  - Each of dogs has its' owner that can be changed.
  - Each person owns any count of dogs (even none).



- Name of association relation – `DogOwning`.
- Role of both classes in the association relation – `owner`, `ownee`.
- Cardinality (multiplicity) of the relation.

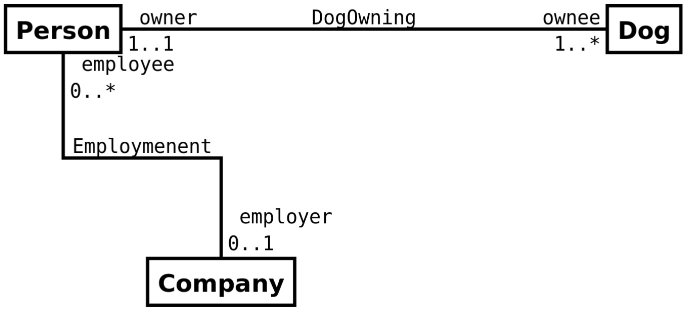
Association Direction



- Arrow shows in which direction is it easy to find other participant(s) of the relation.

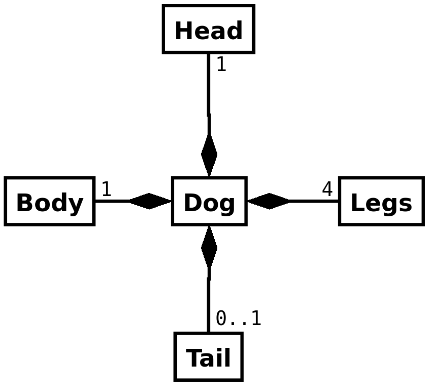
Association with Multiple Classes

- Class can be associated with more classes.
- It has multiple roled.



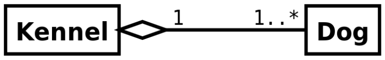
3.3 Composition

Composition



3.4 Aggregation

Aggregation


Kennel
1
1..*
Dog