Unified Modeling Language

Lesson 3: Static View Diagrams

Lesson Objectives

- To understand the following topics:
  - Static View Diagrams
    - Class Diagrams
    - Object Diagrams

3.0. Static View Diagrams
Overview

- Static View Diagrams include:
  - Class Diagrams
  - Object Diagrams

Static View Diagrams:

As seen in the earlier lesson, UML diagrams are classified as:

Dynamic View Diagrams

Static View Diagrams

Physical View Diagrams

Let us begin with the Static View Diagrams.

Static View Diagrams include:

Class Diagrams

Object Diagrams

3.1: Class Diagrams
Features

- Class Diagrams:
  - Class Diagrams define the basic building blocks of a model, namely:
    - types
    - classes, and
    - general material used to construct the full model

Class Diagrams: Features:

Class Diagrams can be used to model classes, and the relationships between classes.

When drawn during the analysis stage, only the names of the classes maybe represented.

During further refinements in the detailed analysis or design stage, details like "attributes" and "services" get added to each class, and are depicted in the Class Diagram.

3.1: Class Diagrams
Functions

- Class Diagrams have the following functions:
  - They describe the static structure of a system.

  - They show the existence of classes and their relationships.
    - Classes represent an abstraction of entities with common characteristics.
    - Relationships may be:
      - Generalization
      - Association
      - Aggregation
      - Composition, or
      - Dependency

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

3.1: Class Diagrams
Uses

- Typical uses of Class Diagrams are:
  - To model vocabulary of the system, in terms of system's abstractions
  - To model collaborations between classes
  - To model logical database schema (blueprint for conceptual design of database)
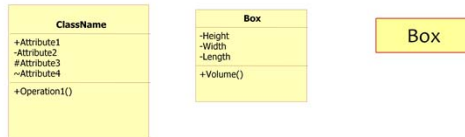
Uses of Class Diagram:

The importance of the Class diagram is that it gives a view of all the classes that are required to make up the system. It also conveys the collaborations that exist between classes to give the system behavior.

3.1: Class Diagrams
Notations for Class

- Class may be represented in any of the following ways
  - Only Class Name is mandatory

ClassName
+Attribute1
-Attribute2
#Attribute3
~Attribute4
+Operation1()

Box
-Height
-Width
-Length
+Volume()

Box

Notations for Class:

Classes are denoted as rectangles, with compartments for name, attributes, and operations. There is optionally a last compartment that can be used for specifying responsibilities, variations, business rules, etc.

The name is the mandatory part. Other compartments may be included based on the amount of details required to be communicated. The representations of classes that do not have all compartments are known as "elided notations for class".

3.1: Class Diagrams

Notations for Class (Contd…)

- Class Visibility signifies how information within class can be accessed.

Symbol	Meaning
+	Public
_	Private
#	Protected

Notations for Class: Class Visibility:

Information about visibility of attributes and operations can sometimes be represented by using symbols like + for public, - for private, or # for protected.

These symbols may vary from tool to tool.

3.1: Class Diagrams
Association Relationship - Features

- In Association:
  - Name indicates relationship between classes.
  - Role represents the way classes see each other.
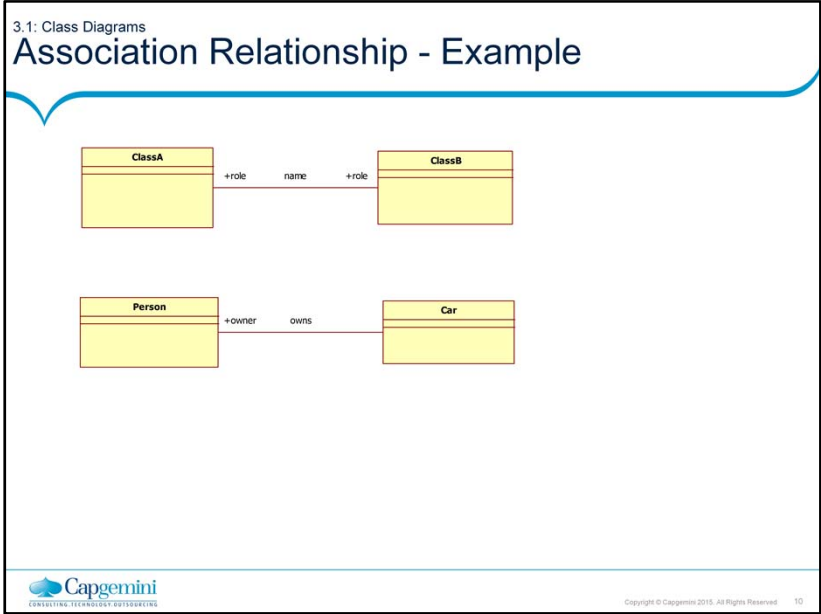
Relationships: Association:

Associations may be characterized by the following:

> Name: The name signifies purpose of association, and is written along with the line indicating association, role and direction of association.

> Role: In case there are specific roles played by classes in the association, then it is indicated by the role name, which is written near the class.

> Arrow: Arrows may be used to indicate whether the association is uni-directional or bi-directional. Absence of arrows implies that no inferences can be drawn about the navigability.

The example in the slide shows an association relationship between a class Person and a class Car. The class Person plays the role of an owner.

3.1: Class Diagrams
Association Relationship - Example

ClassA
+role   name   +role   ClassB

Person
+owner   owns   Car

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved   10

3.1: Class Diagrams

Relationships - Features

- Aggregation and Composition:
  - The following Class Diagram, possessing Composition and Aggregation, displays:
    - Aggregation as indicated by a hollow diamond.
    - Composition as indicated by a filled diamond.
    - Diamond as pointing towards the "whole" class or the aggregate.

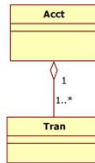Relationships: Aggregation and Composition:

Composition and Aggregation are modeled with filled diamond and hollow diamond, respectively, on the "Whole" part.

Roles and multiplicity, if required, can be mentioned here, as well. Typically they are done for the "Part" part of the "Whole" part.
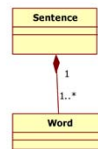
3.1: Class Diagrams

Relationships - Examples

Aggregation	Composition

| Acct |
| Tran |

| Sentence |
| Word |

Examples of Aggregation and Composition:

In the examples shown in the slide,

> the relationship between a Sentence and a Word is represented as a "Composition" (Word is a part of a sentence).

> the relationship between Account and Transaction is represented as an "Aggregation".

3.1: Class Diagrams
Definition of Multiplicity

- Multiplicity:
  - Multiplicity indicates the "number of instances" of one class linked to "one instance" of another class.

Symbol	Meaning
1	Exactly one
0..1	Zero or one
*	Many
0..*	Zero to many
1..*	One to many

Multiplicity:

Multiplicity attached to a class denotes the possible cardinalities of objects of the association.

For example: The above figure depicts that "One company has one or more departments, and a department is associated with one company".

Multiplicity values can be indicated in association, aggregation, and composition relationships.

3.1: Class Diagrams
Association Class Relationship - Features

- Association Class:
  - An Association Class is a class that has properties of both an "association" and a "class".
  - It is required when properties result from unique combination of two classes.
  - For example:

Association class:

An Association class is a class required as the result of association between two classes.

For example:

> The Prize class is a result of association between the Horse class and the Race class.
>
> For each Horse placed in a Race there is a prize.
>
> The amount of prize depends on the race.
>
> The Prize class could not be associated with the Horse class alone because a Horse might have many Prizes, and the relationship between the Prize and Race would be lost.
>
> Similarly, Prize class cannot be associated with Race class alone because a Race has many Prizes, and the relationship between the Prize and the Horse would be lost.

Similarly, result of a student (in terms of marks in assignments, test, and grade) in a course is a unique combination of an individual student, and a particular course. So we can have an association between Student and Course Classes, with Result being an Association Class.

3.1: Class Diagrams
Dependency - Features

- Dependency:
  - Dependency is a "using" relationship within which the change in the specification of one class may affect another class that uses it.
  - For example:

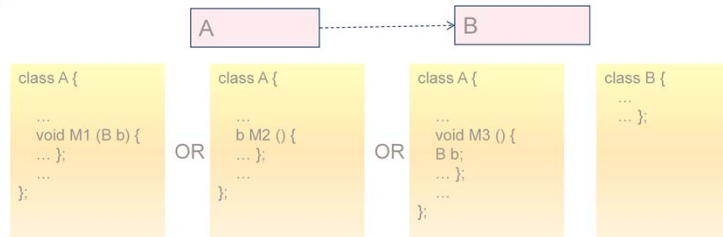| Window		Event
+HandleEvent()	---- - ->	

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
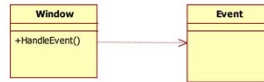
Dependency:

The dependencies are denoted as dashed arrows with arrow head pointing to the independent element.

In the example shown in the slide, the structure and behavior of the Window Class is dependent on the structure and behavior of the Event Class.

3.1: Class Diagrams
What does Dependency Translate to in Code?

A ┄┄┄┄┄┄┄┄▷ B

```
class A {
...
    void M1 (B b) {
    ... };
...
};
```
OR
```
class A {
...
    b M2 () {
    ... };
...
};
```
OR
```
class A {
...
    void M3 () {
    B b;
    ... };
...
};
```
```
class B {
...
... };
```

- Dependencies can translate to one of the following:
  - Instance of Class B is a parameter for method(s) of Class A.
  - Object or reference of Class B is returned by method(s) of Class A.
  - Instance of Class B is a local variable in method(s) of Class A.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING   Copyright © Capgemini 2015. All Rights Reserved   16

UML Relationships: What does Dependency translate to in code?

In a dependency relationship, an instance of the independent class (B) will be used in the dependent class (A) in one of the following manners:

> Instance of B can a parameter to one or more methods of Class A.

> Instance of B can be returned by one or more methods of Class A.

> Instance of B can be a local variable in one or more methods of Class A.

Note that dependency is non-structural, that is, instance of Class B does not come as an attribute of Class A and hence not part of the structure of Class A. Class A is aware of existence of Class B only when the concerned method is called. The relationship is temporary in nature too…once the method invocation is completed, Class A need not maintain information about Class B.

3.1: Class Diagrams
Generalization - Features

- Generalization:
  - Generalization indicates relationships between super-class and sub-class.
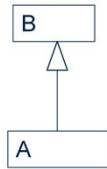 - For example:
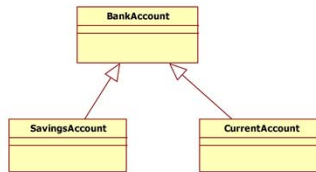
Relationships: Generalization:

Generalizations are denoted as "paths" from specific elements to generic elements, with a hollow triangle pointing to the more general elements.

3.1: Class Diagrams

What does Generalization Translate to in Code?

B

A

```
class B {

...

...

... };
```

```
class A extends B {

...

...

... };
```
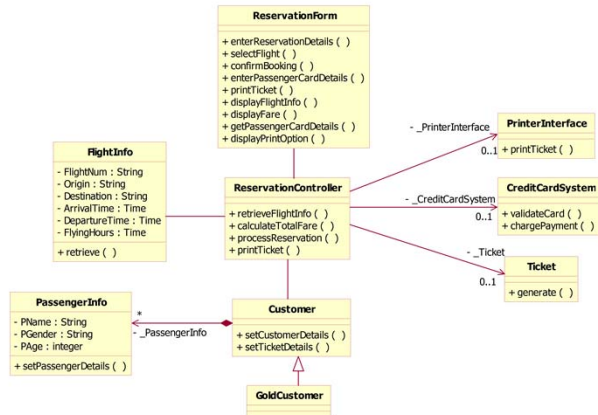
- Generalization entails using the language constructs to implement inheritance relationship.

UML Relationships: What does Generalization translate to in code?

Object oriented languages provide language constructs for implementing inheritance relationship. This will be used for coding generalization.

3.1: Class Diagrams
Example of Class Diagrams

ReservationForm
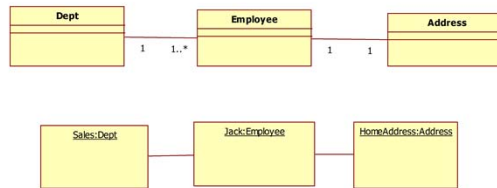+ enterReservationDetails ( )
+ selectFlight ( )
+ confirmBooking ( )
+ enterPassengerCardDetails ( )
+ printTicket ( )
+ displayFlightInfo ( )
+ displayFare ( )
+ getPassengerCardDetails ( )
+ displayPrintOption ( )

- _PrinterInterface

PrinterInterface
0..1 + printTicket ( )

FlightInfo
- FlightNum : String
- Origin : String
- Destination : String
- ArrivalTime : Time
- DepartureTime : Time
- FlyingHours : Time
+ retrieve ( )

ReservationController
+ retrieveFlightInfo ( )
+ calculateTotalFare ( )
+ processReservation ( )
+ printTicket ( )

- _CreditCardSystem

CreditCardSystem
0..1 + validateCard ( )
+ chargePayment ( )

- _Ticket

Ticket
0..1 + generate ( )

PassengerInfo
- PName : String
- PGender : String
- PAge : integer
+ setPassengerDetails ( )

*
- _PassengerInfo

Customer
+ setCustomerDetails ( )
+ setTicketDetails ( )

GoldCustomer

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved   19

How do you interpret this diagram?

Object Diagrams:

The Object Diagrams describe an instantiation of Class Diagram at a particular instance of time. They may be used to explore different configurations of objects. These configurations when combined can be generalized into relevant Class Diagrams.

Object Diagrams provide a snapshot of the instances in a system, and the relationships between the instances. It is a structural diagram that shows the instances of the classifiers in models.

In the example shown in the slide – Sales, Jack, HomeAddr are instances of the classes like department, person, and address respectively. Object Name may be omitted. That is to say, if there is only a colon followed by Class Name (with underlining) such an object is called an "anonymous object".

Objects too may have other compartments as defined by their classes. However, usually attributes (along with values) are mentioned and operations are not mentioned.

contd.

3.2. Object Diagrams

Features (Contd…)

- Object Diagrams are different from Class Diagrams
  - This is because many objects of same class may exist in the Object Diagram.

- Object Diagrams can be used:
  - to test Class Diagrams for accuracy
  - to verify system performance at given instance
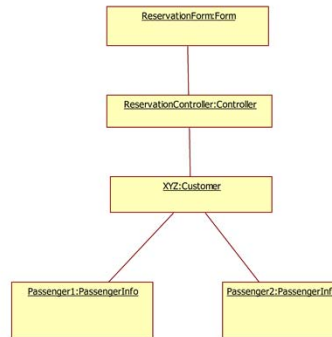 - to optimize performance (especially useful for server objects)

Object Diagrams (contd.):

Depending on the multiplicity values associated with class relationships, there may be different number of objects of the classes that exist in the Class Diagram. "Links" relate the objects. They are instances of relationships of Class Diagrams.

Since Object Diagrams indicate objects of system at particular instance of time, they can be used to verify system performance in terms of memory utilization of objects, and communication links between them. Thought can be given to minimize use of resources by techniques like pooling or queuing, batch processing, etc.

3.2: Object Diagrams
Example of Object Diagrams

ReservationForm:Form

ReservationController:Controller

XYZ:Customer

Passenger1:PassengerInfo   Passenger2:PassengerInfo

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING   Copyright © Capgemini 2015. All Rights Reserved   22

How would you interpret this?

Summary

- In this lesson, you have learnt:
  - Class Diagrams describe the static structure of a system
    - Class Diagrams show the existence of classes and relationship between them like Generalization, Association, Aggregation, Composition, or Dependency
  - Object Diagrams can be used to test the accuracy of Class Diagrams, and to optimize their performance, as well.

Summary

Review Question

- Question 1: A Class Diagram gives information about:
  - A. Attributed defined for a class
  - B. Operations defined for a class
  - C. Logic to be used for an operation of a class

- Question 2: An Object Diagram is unique for an application.
  - True / False

- Question 3: Relationships that you may find on a Class Diagram are ___, ___, ___, ___ and ___.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

24

Review Question: Match the Following

1. **Object Diagram**	A. **Object and Links**
2. **Class Diagram**	B. **Classes and Relationships between then.**