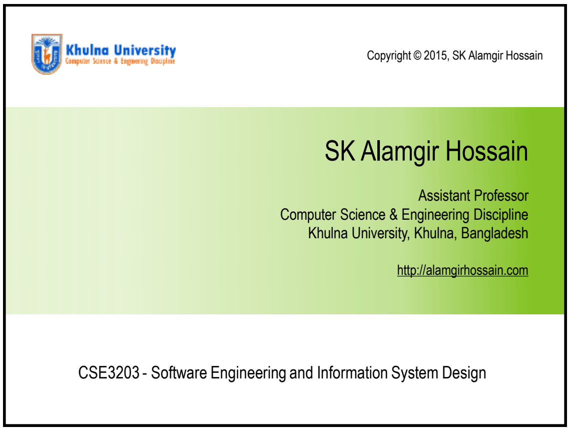
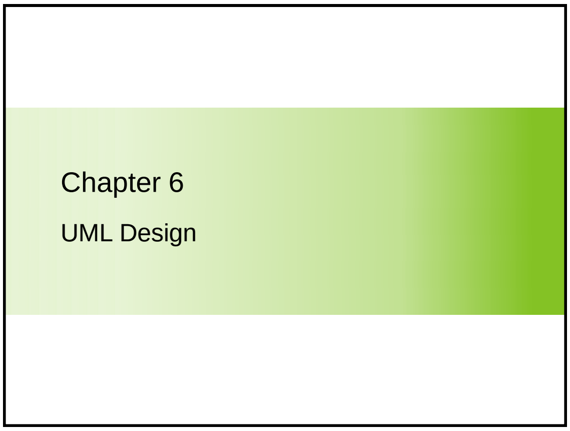


SK Alamgir Hossain

Assistant Professor
Computer Science & Engineering Discipline
Khulna University, Khulna, Bangladesh

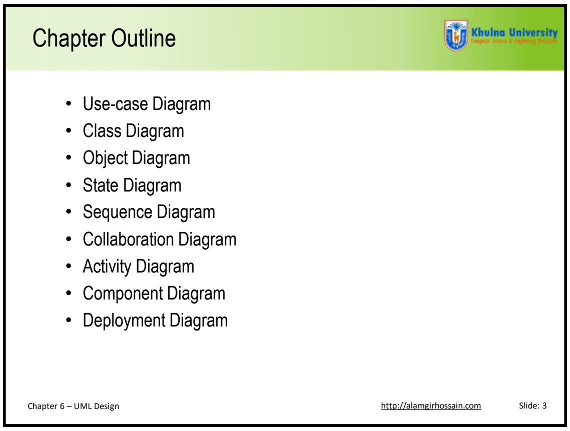
http://alamgirhossain.com

CSE3203 - Software Engineering and Information System Design



Chapter 6

UML Design



Chapter Outline

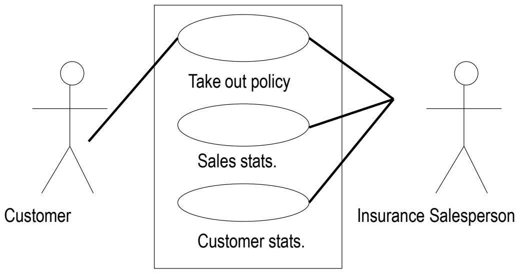
- Use-case Diagram
- Class Diagram
- Object Diagram
- State Diagram
- Sequence Diagram
- Collaboration Diagram
- Activity Diagram
- Component Diagram
- Deployment Diagram

Lecture 11

UML Design

- Use-case Diagram
- Class Diagram
- Object Diagram
- State Diagram
- Sequence Diagram
- Collaboration Diagram
- Activity Diagram
- Component Diagram
- Deployment Diagram

Use Case Diagram (Mainly used in Analysis)



Take out policy

Sales stats.

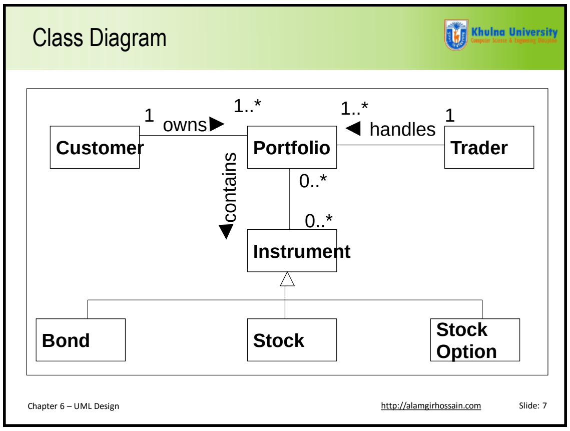
Customer stats.

Customer

Insurance Salesperson

Use-case Notations

- Describe functionality requirements of the system, i.e. functional spec.
- May be described in plain text.
- May be supported by activity diagrams or state diagrams.



Class Diagram

1 owns▶ 1..*

1..* ◀ handles 1

Customer

Portfolio

Trader

contains▼

0..*

0..*

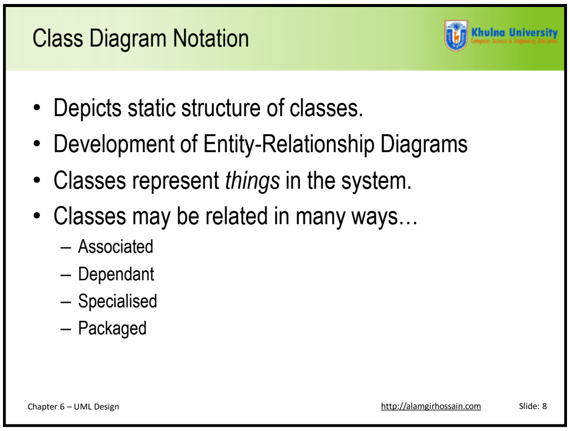
Instrument

Bond

Stock

Stock Option

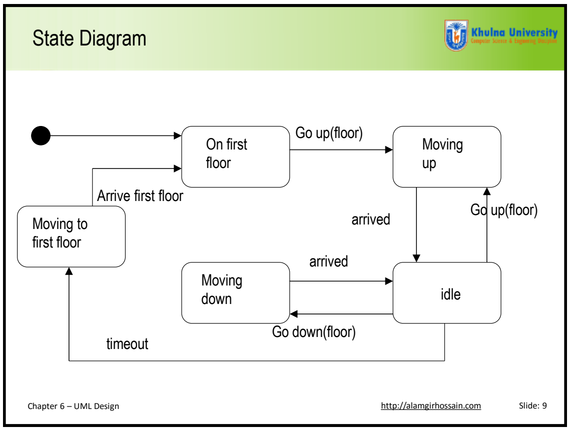
Chapter 6 – UML Design http://alamgirhossain.com Slide: 7



Class Diagram Notation

- Depicts static structure of classes.
- Development of Entity-Relationship Diagrams
- Classes represent *things* in the system.
- Classes may be related in many ways…
  - Associated
  - Dependant
  - Specialised
  - Packaged

Chapter 6 – UML Design http://alamgirhossain.com Slide: 8



State Diagram

On first floor

Go up(floor)

Moving up

Arrive first floor

Moving to first floor

arrived

Go up(floor)

arrived

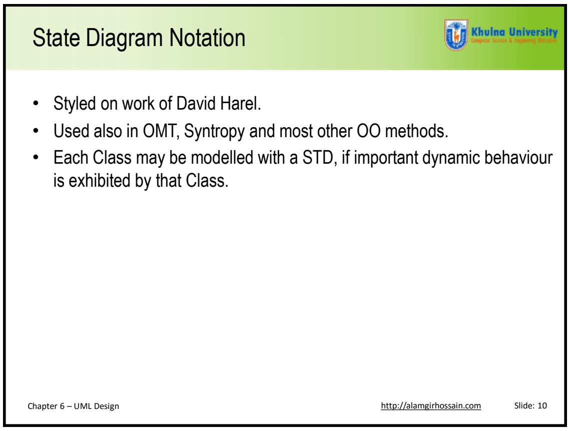
Moving down

idle

Go down(floor)

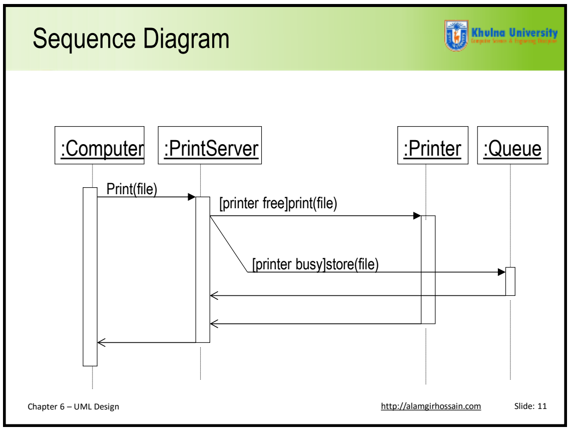
timeout

Chapter 6 – UML Design http://alamgirhossain.com Slide: 9

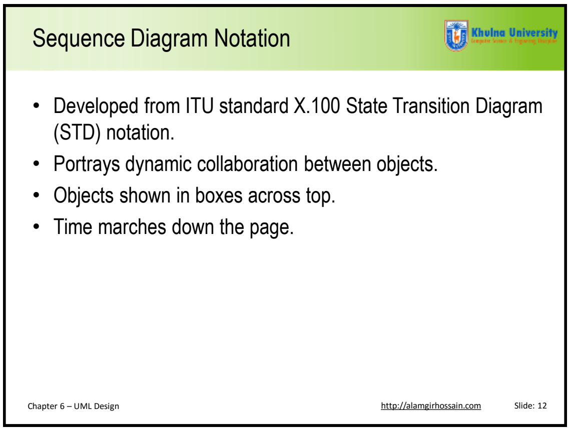


State Diagram Notation

- Styled on work of David Harel.
- Used also in OMT, Syntropy and most other OO methods.
- Each Class may be modelled with a STD, if important dynamic behaviour is exhibited by that Class.



Sequence Diagram



Sequence Diagram Notation

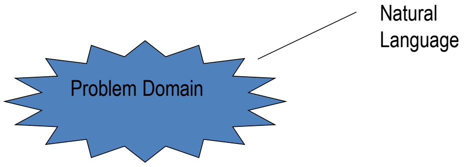
- Developed from ITU standard X.100 State Transition Diagram (STD) notation.
- Portrays dynamic collaboration between objects.
- Objects shown in boxes across top.
- Time marches down the page.

Engineering Process for Allocation of Responsibility

- Process will lay down rules for timing of allocation of responsibilities to classes.
- May use domain analysis, find classes & relationships, then allocate from use cases.
- May find classes from use cases.

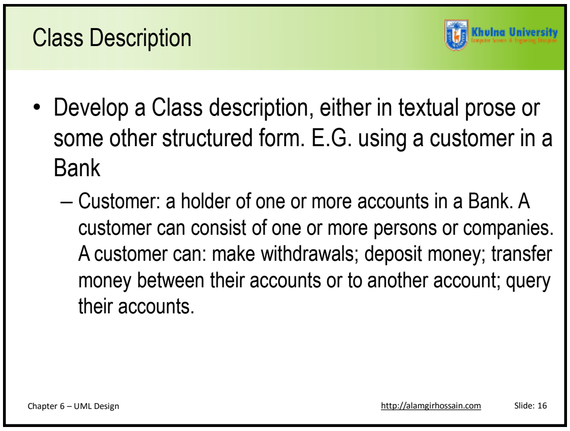
Domain Analysis

- Use Natural Language



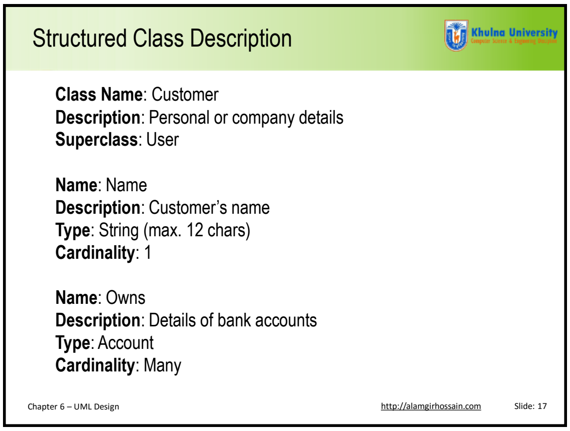
Natural Language

- Nouns suggest **candidate Classes.**
- Not every noun is an Object Class.
  - Some are **attributes** of another Class.
  - Some are irrelevant, outside the scope of the application.
- Verb phrases suggest class associations
  - some relationships are irrelevant (caution).
- Proper nouns suggest Objects of a Class type. Beware of singular nouns.



Class Description

- Develop a Class description, either in textual prose or some other structured form. E.G. using a customer in a Bank
  - Customer: a holder of one or more accounts in a Bank. A customer can consist of one or more persons or companies. A customer can: make withdrawals; deposit money; transfer money between their accounts or to another account; query their accounts.

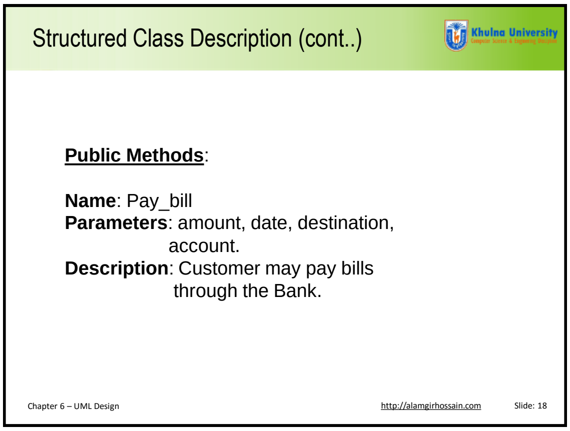


Structured Class Description

Class Name: Customer
Description: Personal or company details
Superclass: User

Name: Name
Description: Customer's name
Type: String (max. 12 chars)
Cardinality: 1

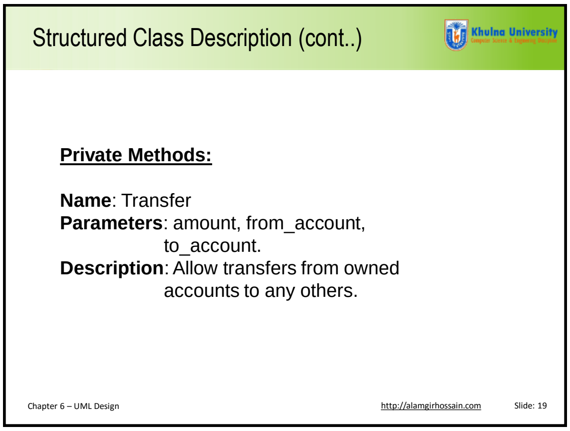
Name: Owns
Description: Details of bank accounts
Type: Account
Cardinality: Many



Structured Class Description (cont..)

Public Methods:

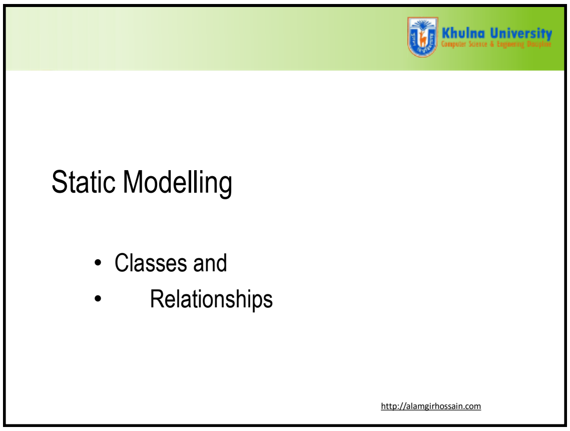
Name: Pay_bill
Parameters: amount, date, destination, account.
Description: Customer may pay bills through the Bank.



Structured Class Description (cont..)

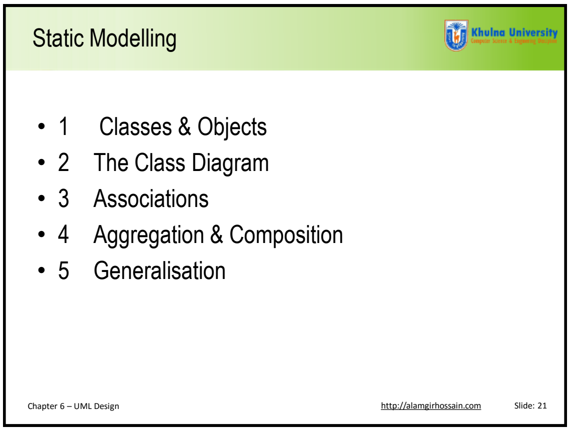
Private Methods:

Name: Transfer
Parameters: amount, from_account, to_account.
Description: Allow transfers from owned accounts to any others.



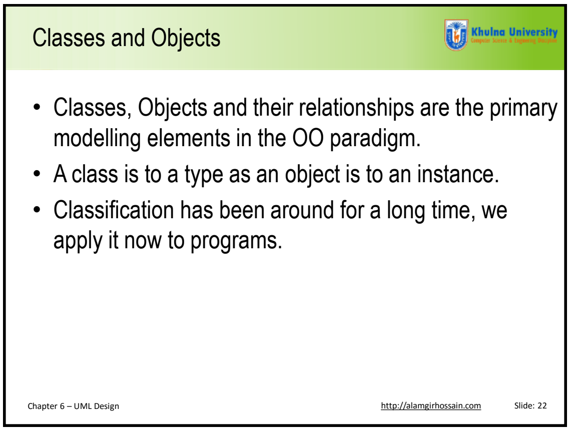
Static Modelling

- Classes and
- Relationships



Static Modelling

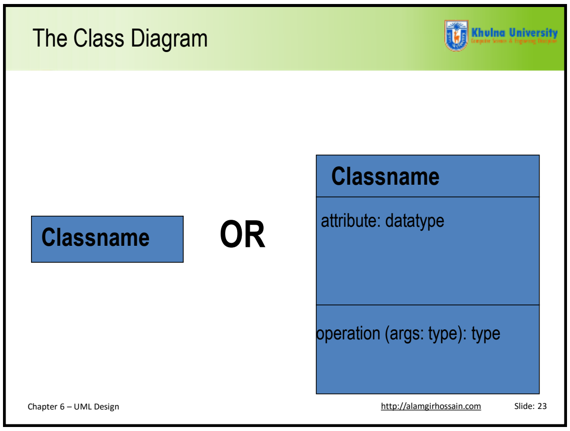
- 1 Classes & Objects
- 2 The Class Diagram
- 3 Associations
- 4 Aggregation & Composition
- 5 Generalisation



Classes and Objects

- Classes, Objects and their relationships are the primary modelling elements in the OO paradigm.
- A class is to a type as an object is to an instance.
- Classification has been around for a long time, we apply it now to programs.

Chapter 6 – UML Design http://alamgirhossain.com Slide: 22



The Class Diagram

Classname

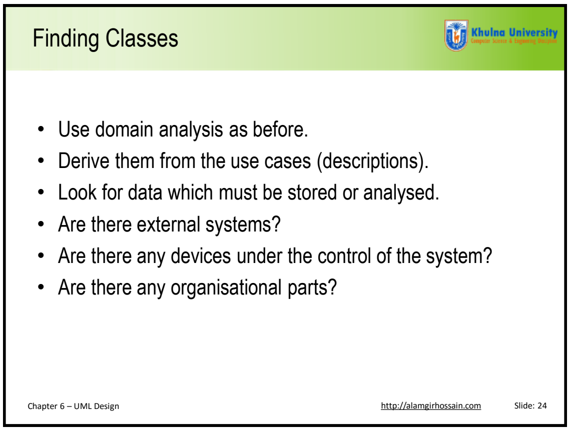
OR

Classname

attribute: datatype

operation (args: type): type

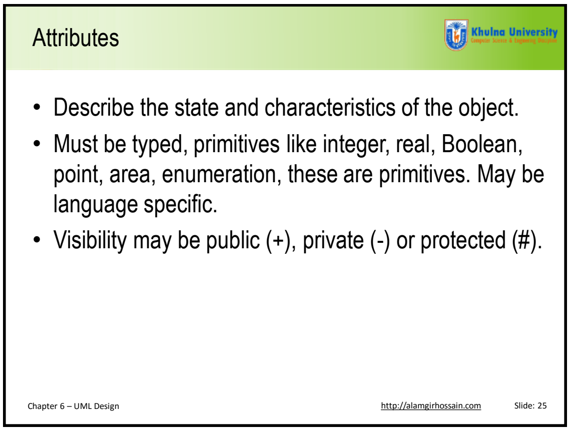
Chapter 6 – UML Design http://alamgirhossain.com Slide: 23



Finding Classes

- Use domain analysis as before.
- Derive them from the use cases (descriptions).
- Look for data which must be stored or analysed.
- Are there external systems?
- Are there any devices under the control of the system?
- Are there any organisational parts?

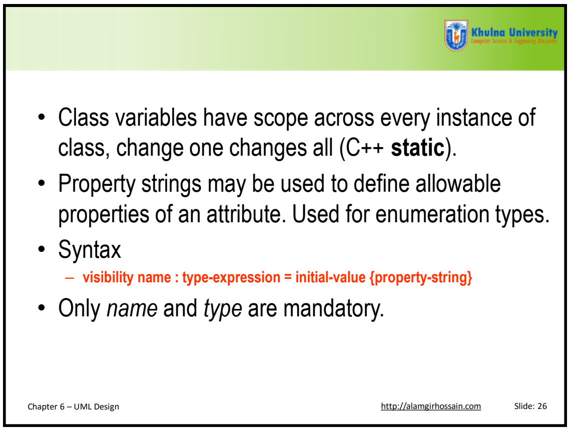
Chapter 6 – UML Design http://alamgirhossain.com Slide: 24



Attributes

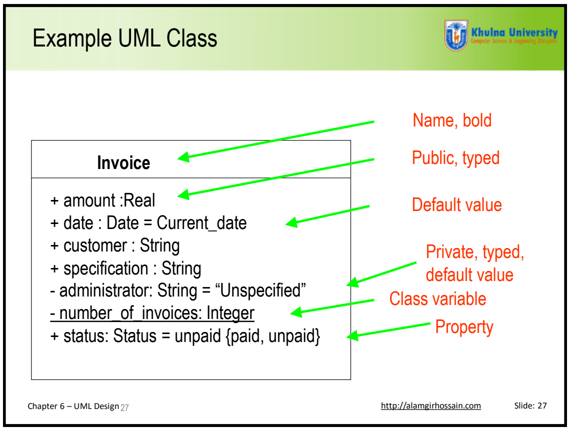
Khulna University

- Describe the state and characteristics of the object.
- Must be typed, primitives like integer, real, Boolean, point, area, enumeration, these are primitives. May be language specific.
- Visibility may be public (+), private (-) or protected (#).



Khulna University

- Class variables have scope across every instance of class, change one changes all (C++ **static**).
- Property strings may be used to define allowable properties of an attribute. Used for enumeration types.
- Syntax
  - visibility name : type-expression = initial-value {property-string}
- Only *name* and *type* are mandatory.



Example UML Class

Khulna University

Invoice

+ amount :Real
+ date : Date = Current_date
+ customer : String
+ specification : String
- administrator: String = “Unspecified”
- number_of_invoices: Integer
+ status: Status = unpaid {paid, unpaid}

Name, bold

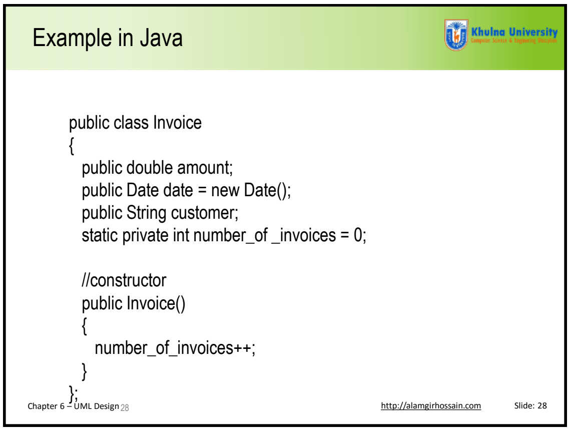
Public, typed

Default value

Private, typed, default value

Class variable

Property

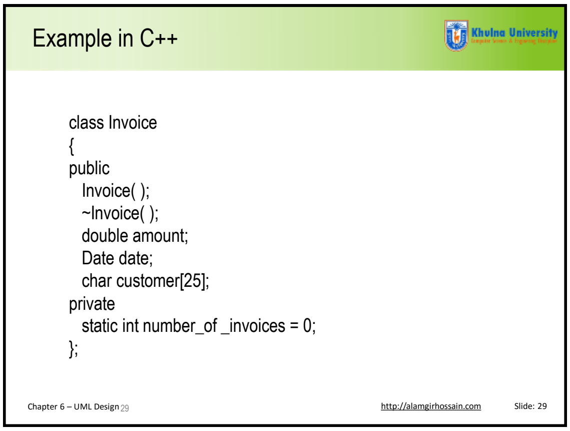


Example in Java

```
public class Invoice
{
   public double amount;
   public Date date = new Date();
   public String customer;
   static private int number_of _invoices = 0;

   //constructor
   public Invoice()
   {
      number_of_invoices++;
   }
};
```

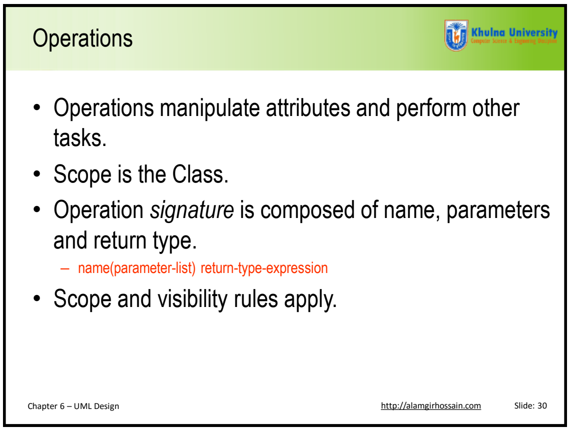
Chapter 6 – UML Design 28 http://alamgirhossain.com Slide: 28



Example in C++

```
class Invoice
{
public
   Invoice( );
   ~Invoice( );
   double amount;
   Date date;
   char customer[25];
private
   static int number_of _invoices = 0;
};
```

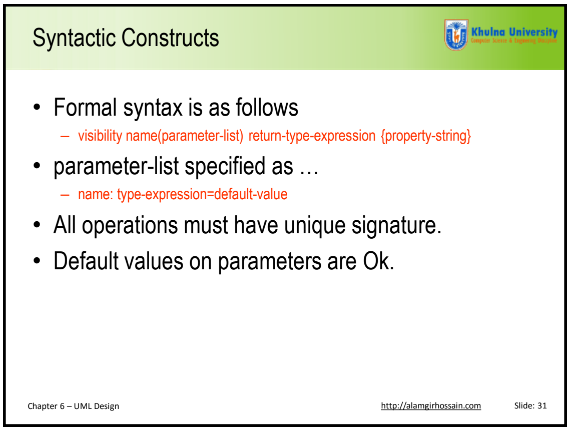
Chapter 6 – UML Design 29 http://alamgirhossain.com Slide: 29



Operations

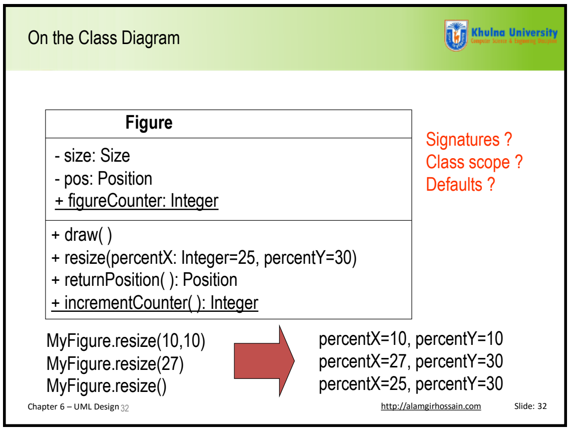
- Operations manipulate attributes and perform other tasks.
- Scope is the Class.
- Operation *signature* is composed of name, parameters and return type.
  - name(parameter-list) return-type-expression
- Scope and visibility rules apply.

Chapter 6 – UML Design http://alamgirhossain.com Slide: 30



Syntactic Constructs

- Formal syntax is as follows
  - visibility name(parameter-list) return-type-expression {property-string}
- parameter-list specified as …
  - name: type-expression=default-value
- All operations must have unique signature.
- Default values on parameters are Ok.



On the Class Diagram

Figure

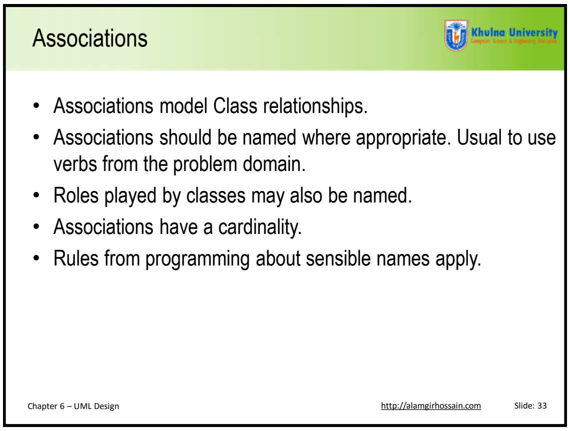
- size: Size
- pos: Position
+ figureCounter: Integer

+ draw( )
+ resize(percentX: Integer=25, percentY=30)
+ returnPosition( ): Position
+ incrementCounter( ): Integer

Signatures ?
Class scope ?
Defaults ?

MyFigure.resize(10,10)
MyFigure.resize(27)
MyFigure.resize()

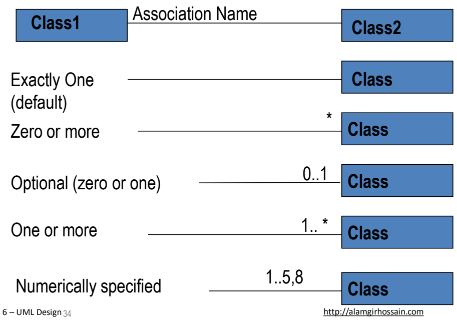
percentX=10, percentY=10
percentX=27, percentY=30
percentX=25, percentY=30



Associations

- Associations model Class relationships.
- Associations should be named where appropriate. Usual to use verbs from the problem domain.
- Roles played by classes may also be named.
- Associations have a cardinality.
- Rules from programming about sensible names apply.

Associations & Cardinality



Class1 Association Name Class2

Exactly One (default) Class

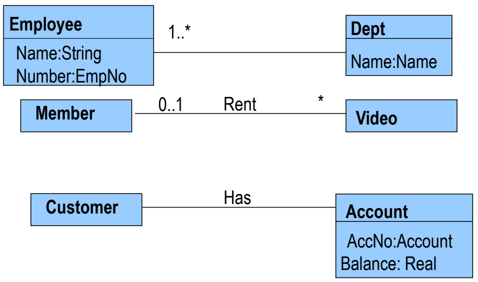
Zero or more * Class

Optional (zero or one) 0..1 Class

One or more 1.. * Class

Numerically specified 1..5,8 Class

Cardinality Examples



Employee
Name:String
Number:EmpNo

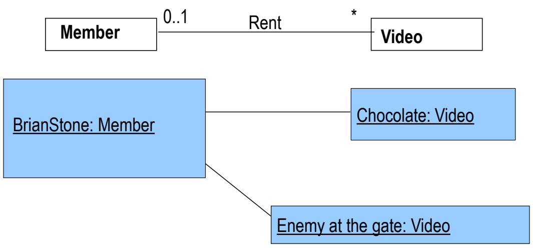
1..*

Dept
Name:Name

Member 0..1 Rent * Video

Customer Has Account
AccNo:Account
Balance: Real

Class & Object Representation



Member 0..1 Rent * Video

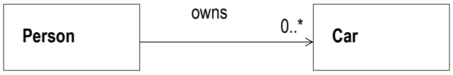
BrianStone: Member

Chocolate: Video

Enemy at the gate: Video

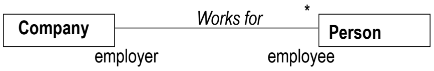
Navigable Associations

- Used to indicate responsibility, later may be translated into pointer mechanism.
- May be bi-directional.



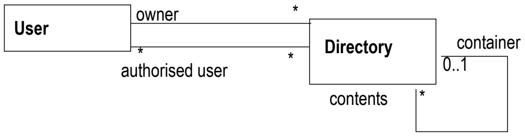
Roles

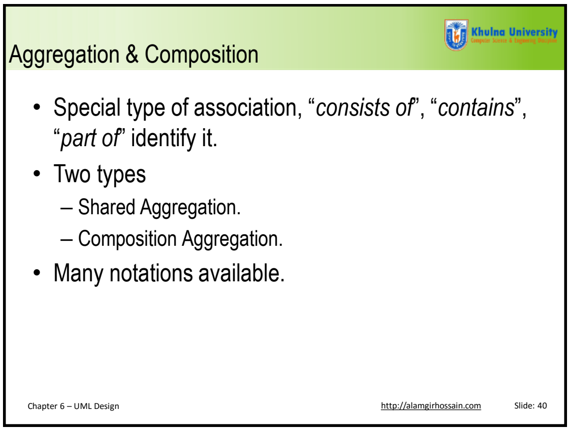
- Useful for indicating context of a class.
- Optional construct.
- Part of the association, not the class



Recursion

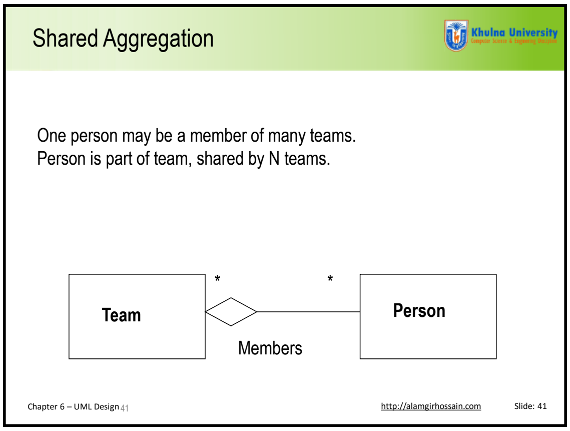
- Self referential construct.
- Complex construct, may not be supported by target language.





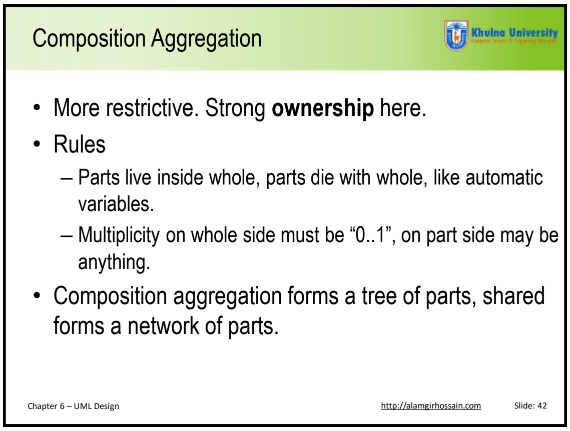
Aggregation & Composition

- Special type of association, "*consists of*", "*contains*", "*part of*" identify it.
- Two types
  - Shared Aggregation.
  - Composition Aggregation.
- Many notations available.



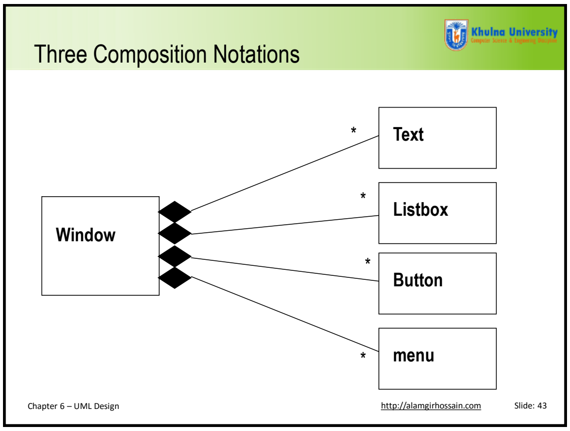
Shared Aggregation

One person may be a member of many teams.
Person is part of team, shared by N teams.



Composition Aggregation

- More restrictive. Strong **ownership** here.
- Rules
  - Parts live inside whole, parts die with whole, like automatic variables.
  - Multiplicity on whole side must be "0..1", on part side may be anything.
- Composition aggregation forms a tree of parts, shared forms a network of parts.



Three Composition Notations

Khulna University

Window

* Text

* Listbox

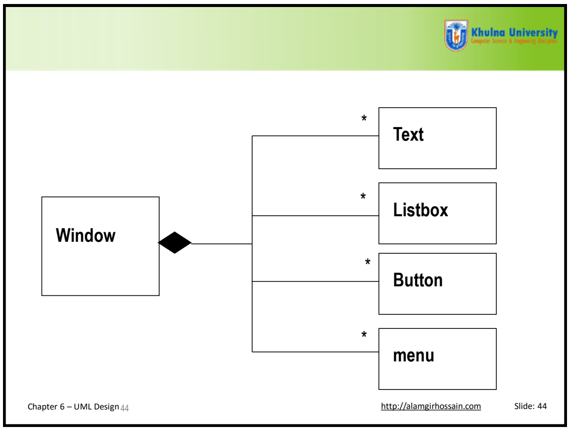
* Button

* menu

Chapter 6 – UML Design

http://alamgirhossain.com

Slide: 43



Khulna University

Window

* Text

* Listbox

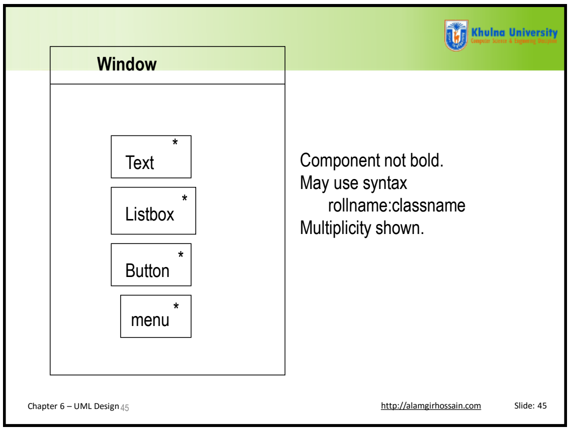
* Button

* menu

Chapter 6 – UML Design 44

http://alamgirhossain.com

Slide: 44



Khulna University

Window

Text *

Listbox *

Button *

menu *

Component not bold.
May use syntax
rollname:classname
Multiplicity shown.

Chapter 6 – UML Design 45

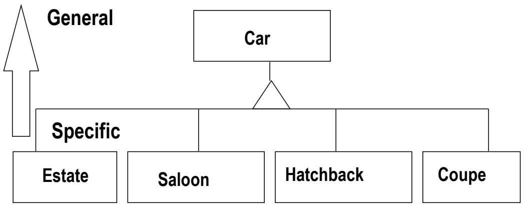
http://alamgirhossain.com

Slide: 45

Generalisation

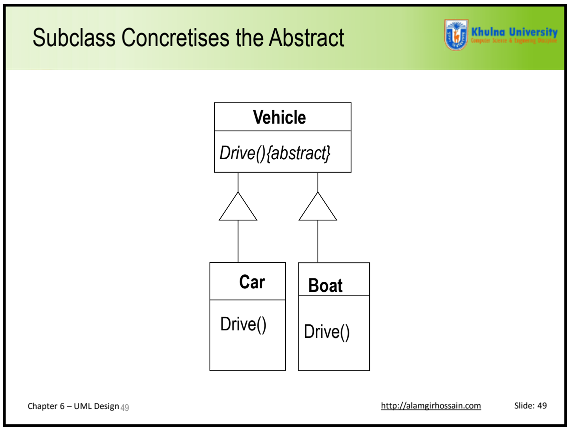
- Generalisation and Inheritance allow sharing of similarities among Classes while also preserving differences.
- Inheritance refers to mechanism of sharing attributes & operations between subclasses and their superclass.
- Default values of attributes & methods for operations may be overridden in subclass.

Example



Normal Generalization

- Subclasses inherit from Superclasses.
- Scope rules apply, public, private and protected are available (+, -, #).
- Abstract classes have no Objects.
- Car class is a good abstract class, denoted with *{abstract}* tag under name in top compartment. Abstract operations are tagged also.



Subclass Concretises the Abstract

Vehicle

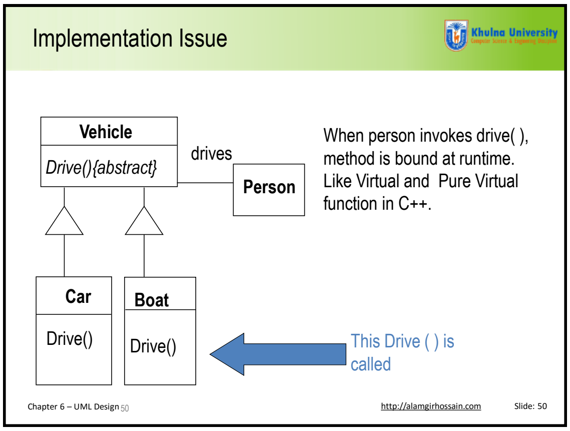
Drive(){abstract}

Car

Drive()

Boat

Drive()



Implementation Issue

Vehicle

Drive(){abstract}

drives

Person

When person invokes drive( ), method is bound at runtime. Like Virtual and Pure Virtual function in C++.

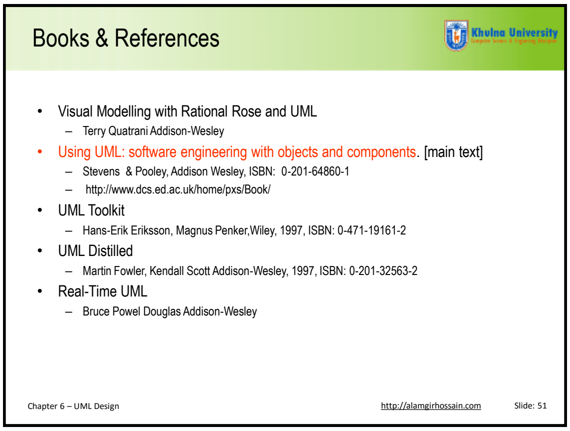
Car

Drive()

Boat

Drive()

This Drive ( ) is called



Books & References

- Visual Modelling with Rational Rose and UML
  - Terry Quatrani Addison-Wesley
- Using UML: software engineering with objects and components. [main text]
  - Stevens & Pooley, Addison Wesley, ISBN: 0-201-64860-1
  - http://www.dcs.ed.ac.uk/home/pxs/Book/
- UML Toolkit
  - Hans-Erik Eriksson, Magnus Penker,Wiley, 1997, ISBN: 0-471-19161-2
- UML Distilled
  - Martin Fowler, Kendall Scott Addison-Wesley, 1997, ISBN: 0-201-32563-2
- Real-Time UML
  - Bruce Powel Douglas Addison-Wesley

THANK YOU