OO System Models
Static Views

UML Class & Object Diagrams
Objective

- Introduces the evolutionary approach for building classes
- Explain how to identify objects and attributes of classes
- Describe the technique of CRC ‘Class Responsibility and Collaborator’
- Explain how classes are related in a class diagram
- Explain generalization, association, aggregation and composition
- Introduce object diagrams
The **Static View** of a system may be described using UML diagrams:

- **UML Class Diagrams**
- **Object Diagrams**
From Use Cases to: Objects, Attributes, Operations (methods) - “evolutionary”
Identifying objects

- Look for **nouns** in the SRS (System Requirements Specifications) document

- Look for **NOUNS** in use cases descriptions

- A **NOUN** may be
  - Object
  - Attribute of an object
Identifying Operations ‘methods’

- Look for verbs in the SRS (System Requirements Specifications) document

- Look for **VERBS** in use cases descriptions

- **A VERB** may be
  - translated to an **operation** or set of operations
  - A method is the code implementation of an operation.
Objects

- **TANGIBLE THINGS**
  - Airplane
  - Book
  - Vehicle
  - Document
  - Worksheet

- **ROLES PLAYED**
  - Employee
  - Customer
  - Doctor
  - Patient
  - End user
  - System administrator

- **ORGANIZATIONAL UNITS**
  - Division
  - Department
  - Section
  - Task force
  - Workgroup

- **DEVICES**
  - Sensor
  - Timer
  - Controller
  - Printer
  - Disk drive
  - Keyboard
  - Display
  - Window
  - Mouse
  - Menu
  - Button

- **INCIDENTS, EVENTS, OR INTERACTIONS**
  - Flight
  - Service call
  - Logon
  - Logoff
  - Contract
  - Purchase
  - Order
  - Payment

- **SITES/LOCATIONS**
  - Warehouse
  - Branch office
  - Factory
  - Retail store
  - Desktop
Objects

An object is a thing:
– student;
– transaction;
– car;
– customer account;
– employee;
– complex number;
– spreadsheet table;
– spreadsheet cell;
– document;
– paragraph;
– GUI Combo box
– GUI button. . . and so on.
Class and Class diagram

- **Class naming**: Use *singular* names
  - because each class represents a generalized version of a singular object.

- **Class diagrams are at the core of OO Eng.**
Class and Class diagram

- Things naturally fall into categories (computers, automobiles, trees...).
- We refer to these categories as classes.

- An object class is an abstraction over a set of objects with common:
  - attributes (states)
  - and the services (operations) (methods)
  provided by each object

- Class diagrams provide the representations used by the developers.
CRC ‘Class Responsibility and Collaborator’

- CRC card

- Class Responsibility:
  - What the class knows: attributes
  - What the class does: services (operations / methods)
CRC ‘Class Responsibility and Collaborator’

- Class Collaboration:
  - Request for **information** from another class (what the other class knows as **attributes**)
  - Request another class to **do some thing** (what the other class does as **operation**)

Software Engineering

OO Models – Class Diagram

Slide 12
# CRC Card

<table>
<thead>
<tr>
<th>CRC Card</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Class name</td>
<td></td>
</tr>
<tr>
<td><strong>Responsibility</strong></td>
<td><strong>Collaboration</strong></td>
</tr>
<tr>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>•</td>
<td>•</td>
</tr>
</tbody>
</table>

What the class knows or does:

Other classes needed to fulfill class a responsibility:

Need for Attribute operation:
## CRC - Class Responsibility

### CRC Card

<table>
<thead>
<tr>
<th>Student</th>
<th>Collaborator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsibility</td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td></td>
</tr>
<tr>
<td>Department</td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td></td>
</tr>
<tr>
<td>Request “Register course”</td>
<td></td>
</tr>
<tr>
<td>Drop course</td>
<td></td>
</tr>
<tr>
<td>Request Schedule</td>
<td></td>
</tr>
</tbody>
</table>

**What the class knows**

**What the class does**
Sometimes a class A has a responsibility to fulfill, but not have enough information to do it.

So class A needs help from another class

See next example
CRC - Class Collaborator

- For example, as you see in students register in courses.
  - To do this, a student needs to know if a spot is available in the course and, if so, he then needs to be added to the course. However, students only have information about themselves (their names and so forth), and not about courses.

  - What the student needs to do is collaborate/interact with the card labeled *Course* to sign up for a course.
  - Therefore, *Course* is included in the list of collaborators of *Student*.
# CRC - Class Collaborator

<table>
<thead>
<tr>
<th>Student</th>
<th>Collaborator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsibility</td>
<td>Course</td>
</tr>
<tr>
<td>ID</td>
<td>(Attribute: availability)</td>
</tr>
<tr>
<td>Name</td>
<td></td>
</tr>
<tr>
<td>Department</td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td></td>
</tr>
<tr>
<td>Check course availability</td>
<td>Course (Operation: increment number of registered student)</td>
</tr>
<tr>
<td>Request “Register course”</td>
<td>Course (Operation: decrement number of registered student)</td>
</tr>
<tr>
<td>Drop course</td>
<td></td>
</tr>
<tr>
<td>Request Schedule</td>
<td></td>
</tr>
</tbody>
</table>

Class **Course** is needed to fulfill class **Student** responsibilities.

Need for **Attribute** and/or **operation**.
Collaboration takes one of two forms:
  • A request for information
  • or a request to do something.

Example Alternative 1:
The card Student requests an indication from the card Course whether a space is available, a request for information.
Student then requests to be added to the Course, a request to do something.

Alternative 2: Another way to perform this logic, however, would have been to have Student simply request Course to enroll himself (Student) into itself (Course). Then have Course do the work of determining if a seat is available and, if so, then enrolling the student and, if not, then informing the student that he was not enrolled.
Class diagrams

- Shows relationship between classes
- A class diagram may show:

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generalization (inheritance)</td>
<td>&quot;is a&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;is a kind of&quot;</td>
</tr>
<tr>
<td>Association (dependency)</td>
<td>does ▶</td>
</tr>
<tr>
<td></td>
<td>&quot;Who does What&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;uses&quot;</td>
</tr>
<tr>
<td>Aggregation</td>
<td>◇</td>
</tr>
<tr>
<td>&quot;has&quot;</td>
<td>&quot;composed of&quot;</td>
</tr>
<tr>
<td>Composition: Strong aggregation</td>
<td></td>
</tr>
</tbody>
</table>
Association, aggregation and composition

- When considering the 3 relationships, association, aggregation and composition,
  - the most **general relationship is association**,  
  - followed by aggregation  
  - and, finally, composition.
Association between classes

Who does What

ORDER # 1043

“IS PLACED BY”

MR. SMITH

“WORKS IN”

ACCOUNTING DEPT.

“CONTAINS”

RED SHIRT SIZE
16/32

“CONTAINS”

401 JEANS SIZE
34 LONG
Mr. Jones has placed no order yet, but there might be many placed over time.

A particular order is placed by Mr. Smith. There can’t be an order without stating who the customer is.

An order contains at least one item, but it could contain many items.
Class diagrams

Abstract & Concrete classes and methods

- **Abstract class**: Has no objects
  - Classes that provide no objects are said to be abstract classes.
  - Its name in *italics*
  - For common attributes/methods of 2 or more child classes
  - Is always a parent class
  - Should not appear below a concrete class in the hierarchy

- **Concrete class**: Has objects (instances)
  - All objects are instances of concrete classes

- **Abstract method**: Has no implementation

- **Concrete method**: Has implementation
Class diagrams

*Abstract & Concrete classes and methods*

Example: Shape

- A super class **Shape** has no implementation of abstract method **Area()**

- A sub class **Square** has an implementation of concrete method **Area()**

- A sub class **Circle** has an implementation of concrete method **Area()**
Inheritance: *is a* “is a kind of”

- *is a* association.
- Child class ‘subclass’ can inherit attributes and operations from parent class ‘superclass’.
- *Example: An inheritance hierarchy in the animal kingdom*
Library class hierarchy

Library item
- Catalogue number
- Acquisition date
- Cost
- Type
- Status
- Number of copies

Methods
- Acquire()
- Catalogue()
- Dispose()
- Issue()
- Return()

Attributes
- Author
- Edition
- Publication date
- ISBN

Is a
- Published item
  - Title
  - Publisher

Is a
- Recorded item
  - Title
  - Medium

Generalisation
- Book
  - Author
  - Edition
  - Publication date
  - ISBN

- Magazine
  - Year
  - Issue

- Film
  - Director
  - Date of release
  - Distributor

- Computer program
  - Version
  - Platform
User class hierarchy

Library user
Name
Address
Phone
Registration #
Register ()
De-register ()

Reader
Affiliation

Borrower
Items on loan
Max. loans

Staff
Department
Department phone

Student
Major subject
Home address
Hierarchy Diagram

(UML notation)

This kind of arrowhead indicates that this relationship is one of subclassing.
## Generalization Naming

<table>
<thead>
<tr>
<th>Generalized class</th>
<th>Specialized class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Superclass</td>
<td>Subclass</td>
</tr>
<tr>
<td>Base class</td>
<td>Derived class</td>
</tr>
<tr>
<td>Parent</td>
<td>Child</td>
</tr>
</tbody>
</table>
Multiple inheritance

- Rather than inheriting the attributes and services from a single parent class, a system which supports multiple inheritance allows object classes to inherit from several super-classes.
- Can lead to semantic conflicts where attributes/services with the same name in different super-classes have different semantics.
- Makes class hierarchy reorganisation more complex.
- Java does not support multiple inheritance.
Example: Multiple inheritance
The talking book

- **Book**
  - Author
  - Edition
  - Publication date
  - ISBN

- **Voice recording**
  - Speaker
  - Duration
  - Recording date

- **Talking book**
  - # Tapes
UML: Associations of regular classes

- Who does what relationship
- When classes are connected together conceptually, that connection is called an association

Association: Who does what
Librarian works in Library

Diagram:
- Librarian
- Employee
- Employer
- Library
- Works in
Associations of regular classes - Who does what

- A manager supervises 1..* employees
- An employee is supervised by 1 manager
Regular & Association classes

- Like a class, an association can have attributes and operations
- You visualize **association class** the same way you show a **regular class**
- Use dotted line to connect **association class** to the association line
Multiplicities of an Association

- Shows the number of objects from one class that relate with a number of objects in an associated class.
UML: Multiplicity

One class can be relate to another in a:

- one-to-one
- one-to-many
- one-to-one or more
- one-to-zero or one
- one-to-a bounded interval (one-to-two through twenty)
- one-to-exactly n
- one-to-a set of choices (one-to-five or eight)
- The UML uses an asterisk (*) to represent *more* and to represent *many*.
Association and Inheritance.

Association:
Who does what
OO: Visibility of attributes or operations

- Visibility: specifies the extent to which other classes can use a given class's attributes or operations.

Three levels of visibility:
- + : public level (usability extends to other classes)
- # : protected level (usability is open only to classes that inherit from original class)
- - : private level (only the original class can use the attribute or operation)
## OO: Visibility

**Ex: Public and private operations in a Hard Disk**

<table>
<thead>
<tr>
<th>HardDisk</th>
</tr>
</thead>
<tbody>
<tr>
<td>+modelName</td>
</tr>
<tr>
<td>+capacity</td>
</tr>
<tr>
<td>+producer</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>+read()</td>
</tr>
<tr>
<td>+write()</td>
</tr>
<tr>
<td>-adjustHeads()</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
Ex: The character hierarchy

- The **Character class** will have **ASCIIcode** and **type** as attributes (type tells the type of the character - normal, italic, bold or underline), and **normal()**, **bold()**, **italic()** and **underline()** as operations. The Character class children will be: **Letter**, **PunctualSign**, **SpecialCharacter** and **Number**.
Ex: Generalization/Specialization Hierarchy

Notation for Motor Vehicles

- MotorVehicle
  - Truck
  - Car
  - Tractor
  - SportsCar
  - Sedan
  - SportUtility

- Trucks, Cars, and Tractors are special types of Motor Vehicles
- Sports Cars, Sedans, and Sport Utilities are special types of Cars
Object Aggregation

- Has-a relationship
- Structural: whole/part
- Peer relationship
  - Whole & parts objects can exist independently
- A special form of association
Object Aggregation: **Peer** relationship

- Whole & parts objects **can exist independently**
- Example: a bank (whole) has customers (as parts)
- Deleting a bank **does not cascade deleting customers**
- Customers can move to another bank

- **Programming**: whole contains an array of parts
Object Aggregation

- Aggregation model shows how classes (which are collections) are composed of other classes.
- Similar to the part-of relationship in semantic data models.
- A line joins a whole to a part (component) with an open diamond on the line near the whole.
Object Aggregation

Example: An aggregation association in the TV Set system

- Every TV has a TV box, screen, speaker(s), resistors, capacitors, transistors, ICs... and possibly a remote control.

- Remote control can have these parts: resistors, capacitors, transistors, ICs, battery, keyboard and remote lights.
Object aggregation

“has”

“composed of”

“part of”
Composition

- A composite is a strong type of aggregation.

- Each component in a composite can belong to just one whole.

- The symbol for a composite is the same as the symbol for an aggregation except the diamond is filled.
Composition – Example 1

- Human's outside:
  Every person has: head, body, arms and legs.

- *A composite association.* In this association each component belongs to exactly one whole.

- Whole & parts objects **can NOT exist independently**
Composition – Example 2

- A bank (whole) has many branches (parts)
- Branches can not exist independently of the whole (parts objects can NOT exist independently)
- Deleting a bank (whole) cascades deleting branches (parts)
- But, if a branch (part) is deleted, the bank (whole) may remain
University Course Enrollment Design

Class Diagram (With Methods)
Example: Class Diagram
Class diagram – Example

Reflexive association

Association: Patient schedules (zero or more) Appointment

Inverse Association: Appointment is associated with only one Patient

Role: Class related to itself

Patient “is primary insurance carrier” of another patient (child, spouse)

Association: Patient suffers (1 or more) Symptom

Inverse Association: Symptom is suffered by (zero or more) Patient