The UML Class Diagram

• Is a static diagram (describes system structure)
  – Combines a number of model elements:
    • Classes
    • Attributes
    • Operations (methods)
    • Associations
    • Aggregations
    • Compositions
    • Generalisations
A UML Class

<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>Attributes</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Operations</td>
</tr>
</tbody>
</table>

Properties of class diagrams:
- Static model;
- Models structure and behaviour;
- Used as a basis for other diagrams;
- Easily converted to an object diagram.
Determining Classes (1/2)

- Is there data that requires storage, transformation or analysis?
- Are there external systems interacting with the one in question?
- Are any class libraries or components being used (from manufacturers, other colleagues or past projects)?
- Does the system handle any devices?
- Does the system model organisational structures?
- Analyse all actor roles.
Determining Classes (2/2)

- **Textual Analysis** *(based on Dennis, 2002)*
  - A *common or improper noun* implies a class
  - A *proper noun or direct reference* implies an object (instance of a class)
  - A *collective noun* implies a class made up of groups of objects from another class
  - An *adjective* implies an attribute
  - A *“doing” verb* implies an operation
  - A *“being” verb* implies a classification relationship between an object and its class
  - A *“having” verb* implies an aggregation or association relationship
  - A *transitive verb* implies an operation
  - An *intransitive verb* implies an exception
  - A *predicate or descriptive verb phrase* implies an operation
  - An *adverb* implies an attribute of a relationship or an operation
UML Class Attributes (1/2)

- Very system dependent
- Describe characteristics of objects belonging to that class
- Can be informative - or confusing
- Has a definite type
  - Primitive (Boolean, integer, real, enumerated, etc.)
  - language specific
  - other classes
  - any user defined type
- Has different visibility, including:
  - public (viewed and used from other classes)
  - private (cannot be accessed from other classes)
UML Class Attributes (2/2)

- Can be given a default value
- Can be given class-scope
- Can list possible values of enumeration
- Directly implementable into most modern programming languages with object-oriented support (e.g. Java)

Attribute syntax:

Visibility name:type=init_value{property_string}
## UML Class Attribute Examples

<table>
<thead>
<tr>
<th>UNIXaccount</th>
<th>UNIXaccount</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ username : string</td>
<td>+ username : string</td>
</tr>
<tr>
<td>+ groupname : string</td>
<td>+ groupname : string = “staff”</td>
</tr>
<tr>
<td>+ filesystem_size : integer</td>
<td>+ filesystem_size : integer</td>
</tr>
<tr>
<td>+ creation_date : date</td>
<td>+ creation_date : date</td>
</tr>
<tr>
<td>- password : string</td>
<td>- password : string</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Invoice</th>
<th>Invoice</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ amount : real</td>
<td>+ amount : real</td>
</tr>
<tr>
<td>+ date : date = current date</td>
<td>+ date : date = current date</td>
</tr>
<tr>
<td>+ customer : string</td>
<td>+ customer : string</td>
</tr>
<tr>
<td>+ specification : string</td>
<td>+ specification : string</td>
</tr>
<tr>
<td>- administrator : string = &quot;unspecified&quot;</td>
<td>- administrator : string = &quot;unspecified&quot;</td>
</tr>
<tr>
<td>- number_of_invoices : integer</td>
<td>- number_of_invoices : integer</td>
</tr>
<tr>
<td>+ status : status = unpaid { unpaid, paid }</td>
<td>+ status : status = unpaid { unpaid, paid }</td>
</tr>
</tbody>
</table>
Public class UNIXaccount
{
    public string username;
    public string groupname = "csai";
    public int filesystem_size;
    public date creation_date;
    private string password;
    static private integer no_of_accounts = 0
    public UNIXaccount()
    {
        //Other initialisation
        no_of_accounts++;
    }
    //Methods go here
};
Public class Figure
{
    private int x = 0;
    private int y = 0;
    public void draw()
    {
        //Java code for drawing figure
    }
};

Figure fig1 = new Figure();
Figure fig2 = new Figure();
fig1.draw();
fig2.draw();
Constraints on Operations

```
BurglarAlarm
isTripped: Boolean = false

PoliceStation
alert (Alarm)
1 station
*
{ if isTripped
then station.alert(self)}

alert (Alarm)
```

Diagram:
- PoliceStation
- BurglarAlarm
- isTripped: Boolean = false
- report ()
- { if isTripped then station.alert(self)}
Association Examples

- Person * Drives * Car
  - Driver
  - Company car
- Person * Employee 1 Car
  - Adult
  - Driver
  - Driver
  - Company car
- Person 1 Married to 1 Person
  - Husband
  - Wife
- Domestic appliance
  - Heater
  - Toaster
  - Radio
  - Turns on
  - Cleans
  - Tunes
- Child
  - Family member
  - Mum
  - Dad
Qualified and "Or" Associations

- Person
  - Plates
  - * Car

- User
  - PID
  - * Process
    - IP-addr
    - * Host

- Item of clothing
  - 1
  - {or}
  - 0..*
    - Male person
    - Female person
Ordered and Ternary Associations

No qualified or aggregation associations allowed in ternary.
Another Ternary Association Example

**Team**

- *team

**Player**

- *goalkeeper

**record**

<table>
<thead>
<tr>
<th>Record</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>goals for</td>
<td></td>
</tr>
<tr>
<td>goals against</td>
<td></td>
</tr>
<tr>
<td>wins</td>
<td></td>
</tr>
<tr>
<td>losses</td>
<td></td>
</tr>
<tr>
<td>ties</td>
<td></td>
</tr>
</tbody>
</table>
Association Classes

- Host
- Network
- Queue
- Printer
- Adapter
- Notary
- Purchaser
- Client
- Contract
- Computer
- Print spooler
- Network adapter
- Real-estate
Association by Aggregation

- Book
  - 1 is part of
  - * has

- Chapter
  - 1 is part of
  - * has

- Crate
  - 1 is in
  - * contains

- Wine bottle
Alternative Notation for Composition Association

Note that association multiplicity is shown within the classes.
Roles in Aggregation

My family

Ernest: Family member
Fiona: Family member

My family

Zoo

Mammal

Bird

Equipment

Falcon

Cage
Abstract Classes

Video
{abstract}

-{abstract} Rental_fee()

VHS

DVD
Abstract Classes and Generalisation Example

Aircraft
{abstract}

- Make
- Seats
- Engine type

Start() {abstract}
land() {abstract}

Jet plane

- Make
- Seats
- Engine type

Start() → Start jet engines
land() → Lower flaps & landing gear

Helicopter

- Make
- Seats
- Engine type

Start() → Start blades
land() → Decrease prop speed
Figure
{abstract}

Position: Pos

Draw() {abstract}

Canvas

Consists of

Electronic

Group

Draw()

Polygon

Draw()

Line

Draw()

Circle

Draw()
Implementing it (e.g. in Java)

abstract public class Figure
{
    abstract public void Draw();
    Pos position;
}

public class Group extends Figure
{
    private FigureVector consist_of;
    public void Draw()
    {
        for (int i = 0; i < consist_of.size(), i++)
        {
            consist_of[i].draw();
        }
    }
}

public class Polygon extends Figure
{
    public void Draw()
    {
        /* something similar to group
         only using lines instead */
    }
}

public class Line extends Figure
{
    public void Draw()
    {
        /* code to draw line */
    }
}

public class circle extends Figure
{
    public void Draw()
    {
        /* code to draw circle */
    }
}
Constrained Generalisations

- **Overlapping**
  - A type of inheritance whereby sharing of common sub-classes by other sub-classes is allowed.

- **Disjoint** (*the default*)
  - The opposite of overlapping.

- **Complete**
  - A type of inheritance whereby the existing sub-classes are said to fully define a given super-class. No further sub-classing may be defined.

- **Incomplete** (*the default*)
  - Further sub-classes can be added later on to more concretely specify a given super-class.
Complete Generalisation

University faculty component

University department
University institute

Person

Man
Woman

{complete}
Expressing Rules in UML

• Rules are expressed using constraints and derivations
  ● Constraints were mentioned earlier (e.g. or-associations, ordered associations, inheritance constraints, etc.)
  ● Derivations are rules governing how entities can be derived (e.g. age = current date - DOB)
Example of Derived Associations

N.B. Relation cardinality is omitted for example clarity
Another Example of a Derived Association

{Supermarket = = (Area > 200 && Category = "dept")}

N.B. Relation cardinality is omitted for example clarity
Example of a Constraint Association

N.B. Relation cardinality is omitted for example clarity
Association Class

Customer \( \rightarrow \) Bill \( \rightarrow \) Supermarket

Bills \( \rightarrow \) Supermarket

Customer \( \rightarrow \) Buys from

Supermarket \( \rightarrow \) Buys from
Class Dependencies

ClassA <<friend>> ClassB

ClassC <<call>> ClassD

ClassC <<refine>>

ClassC combines two logical classes

ClassD

ClassE
Concrete Dependency Example

Diagram Elements

Domain Elements

Graphics Core

«access»

«access»

«access»

«access»

«access»

«access»

«access»

«access»

«access»

«access»
Class Diagram Example

Element

Carbon

Hydrogen

<<covalent>>

C

C

H

<<covalent>>

C

<<covalent>>

C
Instantiation of Class Diagram
(in previous slide)
Try This Yourselves...

- Create a class diagram to represent a arbitrary interconnection of computers

- Create a class diagram to represent a hierarchical directory system in any OS