Introduction to UML
UML diagrams

- Unified Modelling Language (UML)
- Visualise software system, code structure
- Facilitates understanding of software system

An example of UML diagram

```
Frame
   ...

Bounce
   paint()

Oval
   SIZE: int
   x,y: int
   color: Color
   paint()

MovableOval
   deltaX, deltaY: int
   move()```
UML diagrams

An example of UML diagram
UML diagrams

- Use-case diagrams
- Class diagrams
  - Association
  - Aggregation
  - Generalization
- Instance diagrams
- Interaction diagrams
  - Sequence diagram
  - Collaboration diagram

UML diagrams covered in this part
Use-case diagrams

- A high level visualisation of how the system works – use-case modelling
- Built on requirement specifications from discussions between developers and customers, and/or end users

Components

- System
- Actors
- Use case

Use-case diagram shows the interactions between components
System

- Contains components used in a system
- Represented by a labelled box

Banking system
Actor

- “Who/what uses the system”
  - Users of the system
  - Other components that interacts with the system
- Exchanges information with system
  - Send/receive messages
- Initiates/participates in use cases
- A class, not an instance
Actor

- Represented by a labelled stick figure, or a class rectangle

\[
<<\text{Actor}>> \\
\text{Customer}
\]

- Connect Actors by generalisation

Person \rightarrow Customer

OR

Customer
Use cases

- A set of sequence of actions performed by components of a system
- Initiated by an Actor
- Connected to Actor by association
- Represented by a labelled ellipse
- A class, not an instance

Open savings account
Use cases

- Connected to Use cases by generalisation, include, and extend relationships
UML use case diagrams

- Start simple, then refine the details
- Course use case example

![UML Use Case Diagram]

- **Actor**
  - Registrar
  - Student
  - Teacher

- **Use Case**
  - Add course
  - Get course info
  - Register course
  - Enter grade for course

- **Communication**
Exercise

- Identify possible objects from the use case diagram in previous slide

- More examples during lectures
What to do next...

- Identify the objects involved in each use case in the diagram
- Build UML Class diagram to visualise system structure
- Determine inter-object relationships:
  - Association: data & function usage
  - Generalisation: inheritance
  - Aggregation: whole-part relationship
Identifying objects

- Identify key attributes and functions from use cases
- Group closely related attributes
- Group functions by data they act on
Visibility

• May wish to indicate the scope of attributes/methods
  ◆ Default visibility is public (+): can be used by outside classes
  ◆ Protected (#): can be used by descendents of the class
  ◆ Private (-): can only be used within the class itself

• E.g. A more detailed look at the Student class

<table>
<thead>
<tr>
<th>Student</th>
</tr>
</thead>
<tbody>
<tr>
<td>studentID: int</td>
</tr>
<tr>
<td>UPI: String</td>
</tr>
<tr>
<td>Password: String</td>
</tr>
<tr>
<td>registerCourse()</td>
</tr>
<tr>
<td>#changePassword()</td>
</tr>
</tbody>
</table>

Software engineering
Object Relationships

- Identified objects from use case diagram

<table>
<thead>
<tr>
<th>Person</th>
<th>ClassName</th>
</tr>
</thead>
<tbody>
<tr>
<td>name: String</td>
<td>Attributes</td>
</tr>
<tr>
<td>address: String</td>
<td>Operations</td>
</tr>
<tr>
<td>age: int</td>
<td></td>
</tr>
</tbody>
</table>

- How are these objects related???
  Three main ways:
  1. Association
  2. Generalisation
  3. Aggregation
Associations

- Shows how two classes are related
- Represented by a line
- Multiplicity represented by * (≥ 0)
- Specify interval of values
- Default multiplicity = 1
- Common
  - one-to-many
  - many-to-many
  - one-to-ones
 Associations

- A more flexible representation of the relationship between objects
  - A Bounce object uses a Movable object

- Can use labels and arrows to indicate direction
  - A Person borrows a book
Associations

- Association name: verb/verb phrase (function-calls)
- Role name: noun
- Default: has (data)

Zero or more employees *works for* a company

A company *has* a board of directors

Software engineering  Lecture Notes UML 19
Directional associations

- Given an Customer, want to be able to check for the associated PIN, but not vice versa

- A LoginName can be checked with the associated Password, but not vice versa
Reflective associations

- Used for specifying association between interfaces

```
Person

1
supervisor:IManager

* trainee:IEmployee
```
Exercises

- Describe the associations represented by the following diagram. E.g. “A booking is for exactly one passenger”

- Refer to the course example, draw classes linked by associations to represent the relationship between Student, Course, and Lecturer.
  - Assume a student takes at least 1 and at most 4 course
  - A course is lectured by at least 1 lecturer
  - A lecturer must lecture at least 1 course
Inheritance

- **is-a relationship**
  - MovableOval is an Oval
  - Connects a subclass to its superclass
  - Superclass contains attributes/methods common to all its subclasses
  - Subclass contains attributes/methods that are different to its superclass

```
<table>
<thead>
<tr>
<th>Oval</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIZE: int</td>
</tr>
<tr>
<td>x, y: int</td>
</tr>
<tr>
<td>color: Color</td>
</tr>
<tr>
<td>paint()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MovableOval</th>
</tr>
</thead>
<tbody>
<tr>
<td>deltaX, deltaY: int</td>
</tr>
<tr>
<td>move()</td>
</tr>
</tbody>
</table>
```
Generalisation

- Shows inheritance of classes
- Represented by arrows towards the superclass
- Generalise classes with common attributes, associations or operations, where obvious subtyping relationship between parent & children
- Use the ‘is a’ rule, e.g. a Customer is a Person
- A subclass should retain its distinctiveness
- All inherited features must make sense in each subclass
- Recall concepts of abstract classes
Generalisation Example

- Refer to the Bank example
- Customer->Person
  - customer is a person
  - customer inherits person data/functions
- Staff->Person
  - staff is a person
  - staff inherits person data/functions
- Might also have Teller/Manager->Staff etc.
- Avoid over generalisation
Generalisation Example

<table>
<thead>
<tr>
<th>Staff</th>
<th>Person</th>
</tr>
</thead>
<tbody>
<tr>
<td>staffID: int</td>
<td>name: String</td>
</tr>
<tr>
<td>addStaff()</td>
<td>address: String</td>
</tr>
<tr>
<td>findStaff()</td>
<td>age: int</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Teller</th>
<th>Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>counterNumber: int</td>
<td>accountNumber: int</td>
</tr>
<tr>
<td>withdrawCashCounter()</td>
<td>creditLimit: double</td>
</tr>
<tr>
<td>acceptCheque()</td>
<td>withdrawCashATM()</td>
</tr>
<tr>
<td></td>
<td>checkBalanceATM()</td>
</tr>
</tbody>
</table>
Whole-Part

- has-a relationship
- Aggregation
  - A Car has 4 Wheels

```
   Car       Wheel
      ^     /   \     \   /  \
     /       4   /\   /\  \
```

- Composition
  - A Building has a Room

```
   Building     Room
        ^       /   \ \
       /   \   /   / \
```
Aggregation

- Whole-part relationship between objects
- Can be simplified by using associations, isPartOf, or hasParts
- Represented by for composition (strong aggregation)
Aggregation

- Parts are part of the Aggregate
- Aggregate is composed of Parts

- Composition: parts are destroyed if the aggregate is destroyed

\[
\text{Vehicle} \rightarrow \text{VehicleParts}^*
\]
\[
\text{Country} \rightarrow \text{Region}^*
\]
\[
\text{Building} \rightarrow \text{Room}^*
\]

Room cannot exist without a building
Aggregation

- One-to-one aggregation often corresponds to attributes

Employee

<table>
<thead>
<tr>
<th>Addresss</th>
</tr>
</thead>
<tbody>
<tr>
<td>street</td>
</tr>
<tr>
<td>suburb</td>
</tr>
<tr>
<td>city</td>
</tr>
<tr>
<td>country</td>
</tr>
<tr>
<td>postCode</td>
</tr>
</tbody>
</table>

Employee

address: Address
Composition v.s. Inheritance

```java
import java.awt.Point;
public class Circle {
    private Point p;
    private int radius;

    //constructor
    // … code …

    //methods
    public double getX(){return p.getX();}
    public double getY(){return p.getY();}
    public double getRadius(){return radius};
}
```

Circle **has**

- Center point
- radius
Composition v.s. Inheritance

- `getX()`, `getY()` inherit from Point class

```java
import java.awt.Point;
public class Circle extends Point {
    private int radius;

    // constructor
    // ... code ...

    // methods
    public double getRadius() { return radius; }
}
```

Circle *is* Center point
Circle *has* radius
Exercises

- Think about how you would represent a collection of vehicles using generalizations
- Draw a detailed UML class diagram for a school system structure from the following information
  - School: has more than 1 Department and more than 1 Student
  - Department: has at least 1 Staff, including a HOD (Head of Department), offers at least 1 Course
  - Staff: may be a Lecturer, or a Tutor, teaches at least 1 Course
  - Student: attends at least 1 Course and is a part of School
  - Course: is taught by at least 1 Lecturer, has at least 1 student, belongs to a Department
Instance Diagram

- Shows examples (instances) of objects and links
- Also known as an Object diagram
- Models static design/process view of system using instances

<table>
<thead>
<tr>
<th>Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>accountNumber: int</td>
</tr>
<tr>
<td>creditLimit: int</td>
</tr>
<tr>
<td>withdrawCashATM()</td>
</tr>
<tr>
<td>checkBalanceATM()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Helen:Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>accountNumber = 3223589</td>
</tr>
<tr>
<td>creditLimit = 5000</td>
</tr>
</tbody>
</table>

Names are underscored

Assign values to attributes

A class

An instance
 Instance Diagram

- Name of class is underlined and preceded by ‘:’ (can be omitted if obvious)
- Name of instance before colon
- E.g. OOCorp:Company
- Link represented as line (pointer to two objects)

Mary: Employee

Jerry: Employee

OOCorp:Company

or, simplified

:Company
A class diagram is an abstract representation of all instances.

An instance diagram is an example generated from a given class diagram.

Associations represent all possible links between two classes.

Links in instance diagrams do not have multiplicity (*).
Example

- A Customer has at least 1 Account

- Helen has a Savings account

```
<table>
<thead>
<tr>
<th>Helen:Customer</th>
<th>savingsAccount:Account</th>
</tr>
</thead>
<tbody>
<tr>
<td>accountNumber= 3223589</td>
<td>accountType= Savings</td>
</tr>
<tr>
<td>creditLimit= 5000</td>
<td>monthlyFee= 5.00</td>
</tr>
</tbody>
</table>
```
Exercise

- Generate an instance diagram for the CS230 course from the class diagram for the school system
Dynamic Behaviour

- Document dynamic behaviour of software system:
  - Visualise how system is run
  - Show communication between objects, i.e. messages
  - Show the flow of events
Dynamic Behaviour

- Represent using interaction diagrams
  - Sequence diagrams
  - Collaboration diagrams

- Use instances of classes
- Show the passing of parameters
- Show method calls
Function Calling Behaviour

- Idea of an object interacting with another object
- Want to show sequence of calls, perhaps arguments to calls, return values

![Diagram showing function calling behaviour with objects ClassA and ClassB, method calls, and return values]
Sequence Diagrams

- Shows sequence of messages exchanged by objects/actor performing a task
- Emphasizes time ordering of messages
- Events are shown sequentially along a time line
- Illustrates dynamic view of a system (like a snapshot)
Layout and components

- Object arranged from left to right
- Time line: shown in the vertical direction
- Activation Box: shows period where object is active
- Lifeline: the existence of an object over time
- Message represented as labeled arrow between activation boxes of sender, may have arguments and return value

\[
\text{value} := \text{functionName}(\text{arg1}, \text{arg2}, \ldots)
\]
Example

- Sequence of actions for withdrawing cash from an ATM machine

```
addTrans()
updateBalance()
addTrans()
dispenseCash()
getBalance()
checkCRLimit()
```

- Lifeline
- Activation Box
- Self-call
- Called function name
Example

**Course registration**

- Actor requests to register a student.
- CourseSection checks if the student has the prerequisite.
- If the prerequisite is passed, a new Registration object is created and added to the schedule and registration list.
- Course gets the prerequisite.

```plaintext
requestToRegister(aStudent) -> CourseSection
hasPrereq:= hasPassedCourse(prereq)
[hasPrereq] <<create>> Registration
addToSchedule
addToRegList
```
Collaboration Diagrams
Layout and components

- Similar to sequence diagrams
- Show function calling between objects
- Emphasizes structural organisation of object
- Number calls to indicate order of event
- May include arguments
- Basic notation used:
  - Calling sequence number
  - Called function name
  - Arguments
  - Return value
Collaboration Diagrams

- ATM Withdraw cash collaboration diagram

```
1: getBalance( )
2: checkCRLimit( )
3: addTrans( )
4: updateBalance( )
5: addTrans( )
6: dispenseCash( )

:ATMMachine

:Customer

:Transaction

:Account

Withdraw Cash
```
Iterations and conditions

- Can indicate iterations by using * or specifying a valid range in front of the message
  - * : for any number of iterations
  - * [all objects] : performs the operation for all objects concerned
  - * [i=1:n] : performs the operation for n times

- Can specify conditions by prefixing sequence number with a condition clause. Different paths will have same sequence number, but each path must be uniquely distinguishable
  - [x>0] : execute the following operation if x>0
  - [waiting time > 30 sec]: execute the following operation if waiting time exceeds 30 seconds
Iterations and conditions

- Example: A bank customer inserts bank card into an ATM to perform a transaction

```
Customer

:ATMMachine

:Card

insertCard() \rightarrow requestPIN()

*\[i=1:4\]: enterPINDigit() \arrow{dashed}

checkPIN() \rightarrow [verified=true] Proceed with transaction…

[verified=false] Do something else… E.g. ATM eats the card
```
Example

sequence diagram for making a hotel reservation

If a room is available for each day of the stay, make a reservation and send a confirmation.
Example

collaboration diagram for making a hotel reservation
Exercise

- Convert the course registration sequence diagram to the equivalent collaboration diagram
Detailed Descriptions

- NOT sufficient to draw lots of diagrams. Documentation is still necessary.

- Documentation:
  - explain each diagram, object in diagrams
  - explain each object attribute & operation
  - explain associations, function calls
References for UML

Books:
- Unified modeling language user guide: Grady Booch, James Rumbaugh, Ivar Jacobson
- UML toolkit: Hans-Erik Eriksson, Magnus Penker

Links
- UML™ Resource Page
  http://www.uml.org/
- ArgoUML
  http://argouml.tigris.org
What next??

- Now have high-level “specification” for system data & functions
- Need more info in order to implement system:
  - “software architecture” to use
  - user interface vs. data management vs. processing objects
  - data structures, data management
  - detailed OOD class/collaboration diagrams
  - detailed specifications of all design-level data/functions