What is the UML

Premise

*Software systems are complex. We need simpler views of them in order to master that complexity.*

UML is a language for visual modelling.

- Visual modelling is one way of creating accessible abstractions of complex systems.
- UML is a visual language — follows the tradition of Booch notation and OMT.
- UML supports OO analysis and design.

Use of UML

- In analysis and specification phases to model
  * real-world objects and classes, situations, and processes (e.g., business processes).
  * existing software components.
  * interactions between planned software and the above.
- In design phase to model internal components and processes.
- To document legacy systems.
Diagrams of UML

• Class diagrams – classes and packages, their properties, relationships.
• Object diagrams – snapshots of objects and their relationships.
• Use-case diagrams – use cases, actors, relationships.
• Sequence diagrams and Collaboration diagrams – typical sequences of events (e.g., calls).
• Statechart diagrams – finite state machines.
• Activity diagrams – algorithms / data-flow.
• Component diagrams – implementation components (e.g. source & object files)
• Deployment diagrams – deployment of components on computers.
Diagrams shows

- 6 classes
- 3 inheritance relationships
- 2 has-a relationships.
Supplying information about a class

Each class is displayed as a box with 3 or more parts:

- **<<stereotype>> Name.** Stereotypes are used to identify classes that are used in stereotypical ways, e.g. interfaces, abstract classes, actors (agents outside system), exceptions, etc. The Name is the name.
- **Attributes.** (A.k.a. Fields / data members). This class has one.
- **Operations.** (A.k.a. Method signatures, function members).
- **Other parts as you please.** E.g., responsibilities

Operations and attributes are marked according to visibility.
We can model dependance
How to do cyclic calling without cyclic dependance.
Class relationships

- Is-a (specialization): Every D is an M. Class D specializes class M. Class D inherits from class M.
  In C++ we say D derives from M. In Java D extends M.

```
D <-[ ]-> M
```

Note that class D depends on M.

- Realizes. D implements interface M. Special case of above for interfaces.

```
D <-[ ]-> <<Interface>> M
```

or lollypop notation:

```
D --- M
```

---

The Uni...
• Knows-a (association): Every D can (potentially) easily find an M.

In C++ (or Java) D might have a data member (field) that is a pointer to an M.

![Diagram showing association between D and M]

In the above diagram the D object knows 0 of more M objects. In C++ you might have a data member that is a vector of pointers to M objects.

Use a two way arrow if the M object can find the D object that can find it.

Use no arrow if there is an association, but you don’t want to imply that either can find the other.

Usually (with the arrow) D depends on M.

• Has-a (aggregation): Every D has an M’s.

This is a special case of “knows-a”. Use it when the lifetimes are coincident; i.e. creating a D object creates the M object and destroying the D object destroys the M object.
In C++, D might have a private data member of type M called `name`, or D might have a pointer to an M object that is set with `new` when a D is constructed and sent to `delete` when a D is destructed.

- **Depends on:** Use when there is dependance, but none of the above are appropriate.
  
  E.g. Some method D.foo() takes an M as a parameter, returns an M as a result, creates an M, but doesn’t maintain a long term association, or calls a static method of M.

It is good to use a stereotype to describe the type of dependance. E.g.:
Sequence diagrams

Show typical scenarios – not algorithms.

Messages may be sent to self
Sequence diagrams

... can show the interaction of a system with objects outside (specification)
Collaboration Diagrams

Same info as sequence diagram, but in different form

1: loadStr("C++",s)  
2: <<create>>  
3: <<create>>(cppl)  
4: <<create>>

client

: BigApplet

: DisplayManager

: Evaluator

: CPlusPlusLang
State Diagrams

Allows description of finite number of states
A VCR showing substates
Conclusions and Assessment

UML has considerable momentum.

- Lots of books.
- Good industry uptake.

UML is big and expandable.

- It offers something to everyone.
- But it is weak on data flow.
- Assertion language (OCL) is defined, but not widely known and may define semantics of classes better than state or activity diagrams.

Tools

There are several tools that hold models

- Keep diagrams consistent with database.
- Automatic analysis of source code.
- Automatic generation of source code.
- Round-trip engineering.