

Introduction to UML

- What is UML?
 - Unified Modelling Language
 - A modelling language allows the specification, the visualisation and the documentation of a software development process

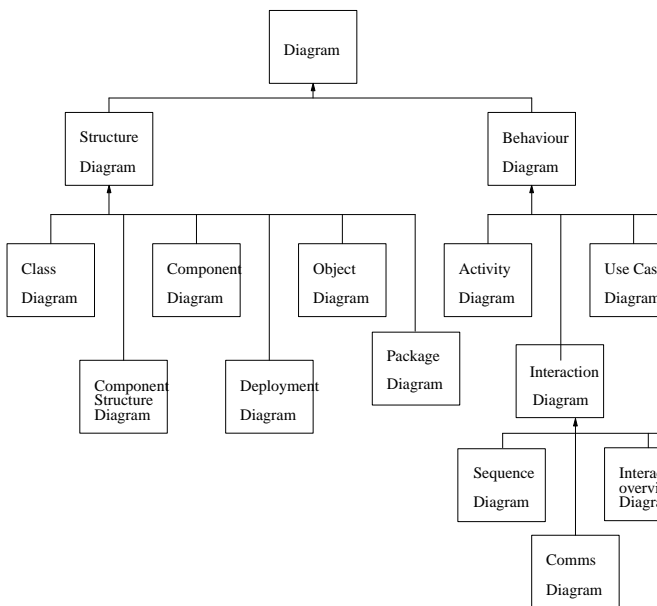


“The soul never thinks without an image”
Aristotle, De Anima

Simplex Veri Sigillum

- simplicity is the hallmark of truth
- complexity continues to have a morbid attraction
- When you give for an academic audience a lecture that is crystal clear from alpha to omega, your audience feels cheated and leaves the lecture hall commenting to each other: “That was rather trivial, wasn’t it?”
 - The sore truth is that complexity sells better
Edsger W.Dijkstra, EWD898

Overview of UML



Advantages of UML

- UML is a useful method of visualisation and documenting software systems design
- uses object oriented design concepts and it is programming language independent
- UML is a popular technique for documenting and modelling a system
- can model a systems structure and behaviour

Disadvantages of UML

- open source projects and free software projects by and large ignore UML
- UML diagrams do not run and they require time to create and maintain
 - you need to keep the diagrams synchronised with your code
- you cannot represent every condition in a diagram
- poor for distributed systems
 - says nothing about where the software resides
 - server process or client process etc

Alternatives to UML

- E-R diagrams for database ER structure
- Petri Nets (often used to design distributed systems)
- SysML for systems engineering applications
- MASCOT for real-time systems
- again these rarely feature in open source/free software projects

The class diagram

- is the main building block of object oriented modelling
- it is used both for general conceptual modelling of the systematics of the application, and for detailed modelling translating the models into programming code
- the classes in a class diagram represent both the main objects
 - the interactions between the application and the classes have to be programmed

An example class (and diagram)

- | Bank account |
|-----------------------------|
| ownersname: String |
| balance: Pounds = 0 |
| deposit (amount: Pounds) |
| withdrawal (amount: Pounds) |

Class diagram sections

- it has three sections
- the top part: the name of the class
 - printed in Bold, centered and the first letter capitalised
- the middle part: attributes of the class
 - left aligned and the first letter is lower case
- the bottom part: the methods or operations for the class
 - left aligned and the first letter is lower case

Class diagram sections

- in the design of a system, a number of classes are identified and grouped together in a class diagram which helps to determine the static relations between those objects
- with detailed modelling, the classes of the conceptual design are often split into a number of subclasses

Class diagram sections

- in order to further describe the behaviour of systems, these class diagrams can be complemented by a state diagram or a UML state machine

Visibility

- the visibility of a class member is defined by the following symbols
 - which are placed before the member name

Symbol	Visibility
+	Public
-	Private
#	Protected
/	Derived (can be combined with one of the others)
-	Static
~	Package

Example

■

Bank account
-ownersname: String -balance: Pounds = 0
+open (name: String) +deposit (amount: Pounds) +withdrawal (amount: Pounds) +close ()

Advantages of UML

- clearly this allows us to specify classes quickly
 - a tool can also generate the appropriate language bindings
 - C++, Java, C#, Objective C
 - with some work a tool could also produce bindings for: C, Modula-2, Ada
- it also creates a common lexicon between developers

Scope

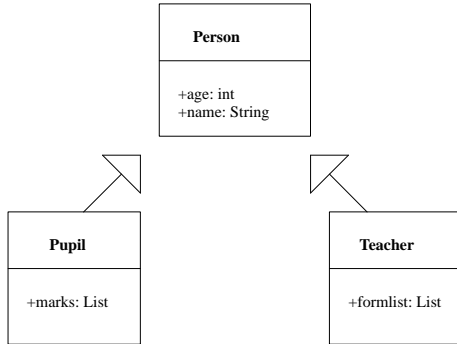
- UML specifies two types of scope for members
 - *instance* and *classifier*
- *classifier* members are commonly recognised as `static` in many programming languages
 - thus scope is the class itself
 - attribute values are the same for all instances
 - are represented by underlining the class name
- *instance* members are scoped to a specific instance
 - attribute values may vary between instances
 - methods or functions may affect the instance's attributes
 - are represented by not underlining the class name (the default)

Class level relationships

- when designing a program it is necessary to consider interaction between classes
- we start to see some of the power of UML when we consider the relationship between classes

Generalisation

- the generalisation relationship is also known as the inheritance or “is a” relationship
- represented by a hollow triangle shape on the superclass end of the line
 - or tree of lines that connects it to one or more subtypes



UML tool

- a UML tool would convert the previous diagram into the following code:

Person.cs

```

public class Person
{
    public int age;
    public String name;
}
  
```

UML tool

- **Student.cs**

```

using Person;

public class Student : Person
{
    List marks;
}
  
```

UML tool

- **Teacher.cs**

```

using Person;

public class Teacher : Person
{
    List formlist;
}
  
```

Conclusion

- there is much more to UML than this!
- notice how you still need to write all the implementation code
- UML class diagrams allow an easier way to consider the design of a multi class program
- you will be using UML next year!