

2 The Unified Modelling Language (UML)

Introduction

This chapter will introduce you to the roles of the Unified Modelling Language (UML) and explain the purpose of four of the most common diagrams (class diagrams, object diagrams, sequence diagrams and package diagrams). Particular emphasis will be placed on class diagrams as these are the most used part of the UML notation.

Objectives

By the end of this chapter you will be able to...

- Explain what UML is and explain the role of four of the most common diagrams,
- Draw class diagrams, object diagrams, sequence diagrams and package diagrams.

The material covered in this chapter will be expanded on throughout later chapters of the book and the skills developed here will be used in later exercises (particularly regarding class diagrams).

This chapter consists of six sections:-

- 1) An introduction to UML
- 2) UML Class Diagrams
- 3) UML Syntax
- 4) UML Package Diagrams
- 5) UML Object diagrams
- 6) UML Sequence Diagrams

2.1 An Introduction to UML

The Unified Modelling Language, UML, is sometimes described as though it was a methodology. It is not!

A methodology is a system of processes in order to achieve a particular outcome e.g. an organised sequence of activities in order to gather user requirements. UML on the other hand a precise diagramming notation that will allow program designs to be represented and discussed. As it is graphical in nature it becomes easy to visualise, understand and discuss the information presented in the diagram. However as the diagrams represent technical information they must be precise and clear – in order for them to work therefore there is a precise notation that must be followed.

As UML is not a methodology it is left to the user to follow whatever processes they deem appropriate in order to generate the designs described by the diagrams. UML does not constrain this – it merely allows those designs to be expressed in an easy to use, but precise, graphical notation.

2.2 UML Class diagrams

Classes are the basic components of any object oriented software system and UML class diagrams provide an easy way to represent these. As well as showing individual classes, in detail, class diagrams show multiple classes and how they are related to each other. Thus a class diagram shows the architecture of a system.

A class consists of:-

- a unique name (conventionally starting with an uppercase letter)
- a list of attributes (int, double, boolean, String etc)
- a list of methods

Maastricht University *Leading in Learning!*

Join the best at the Maastricht University School of Business and Economics!

Top master's programmes

- 33rd place Financial Times worldwide ranking: MSc International Business
- 1st place: MSc International Business
- 1st place: MSc Financial Economics
- 2nd place: MSc Management of Learning
- 2nd place: MSc Economics
- 2nd place: MSc Econometrics and Operations Research
- 2nd place: MSc Global Supply Chain Management and Change

Sources: Keuzegids Master ranking 2013; Elsevier 'Beste Studies' ranking 2012; Financial Times Global Masters in Management ranking 2012

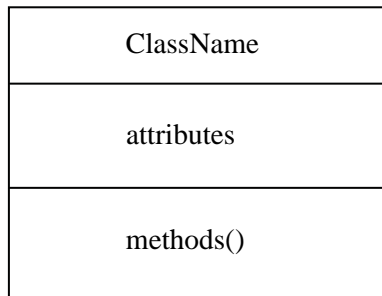
Maastricht University is the best specialist university in the Netherlands (Elsevier)

Visit us and find out why we are the best!
Master's Open Day: 22 February 2014

www.mastersopenday.nl



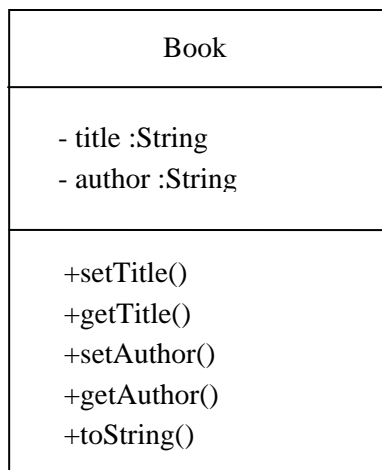
This is shown in a simple box structure...



For attributes and methods visibility modifiers are shown (+ for public access, – for private access). Attributes normally being kept private and methods normally made public.

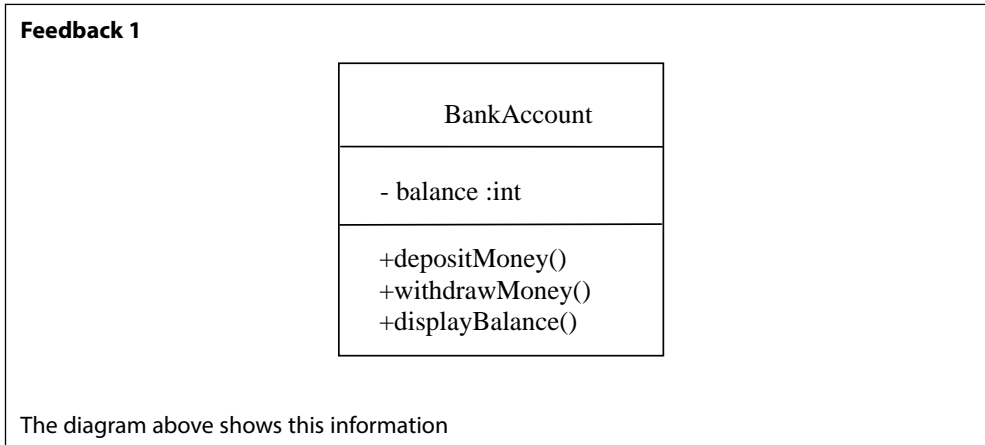
Note: String shown above is not a primitive data type but is itself a class. Hence it starts with a capital letter.

Thus a class Book, with String attributes of title and author, and the following methods setTitle(), getTitle(), setAuthor(), getAuthor() and toString() would be shown as...



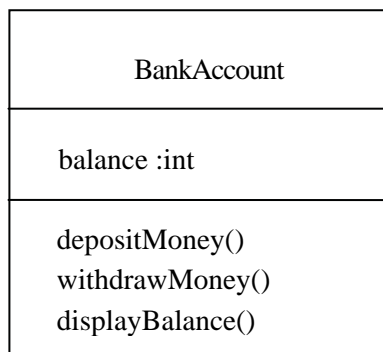
Activity 1

Draw a diagram to represent a class called 'BankAccount' with the attribute balance (of type int) and methods depositMoney() and withdrawMoney(). Show appropriate visibility modifiers.

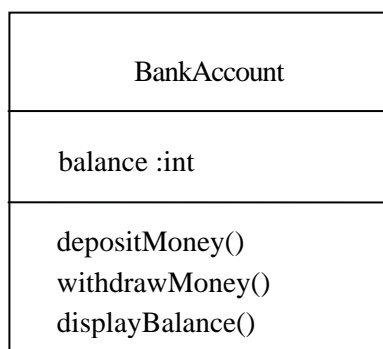


UML allows us to suppress any information we do not wish to highlight in our diagrams – this allows us to suppress irrelevant detail and bring to the readers attention just the information we wish to focus on. Therefore the following are all valid class diagrams...

Firstly with the access modifiers not shown...



Secondly with the access modifiers and the data type not shown...



And finally with the attributes and methods not shown...



i.e. there is a class called 'BankAccount' but the details of this are not being shown.

Of course virtually all Java programs will be made up of many classes and classes will relate to each other – some classes will make use of other classes. These relationships are shown by arrows. Different type of arrow indicate different relationships (including inheritance and aggregation relationships).

In addition to this class diagrams can make use of keywords, notes and comments.

As we will see in examples that follow, a class diagram can show the following information:-

- Classes
- attributes
- operations
- visibility

> Apply now

REDEFINE YOUR FUTURE
AXA GLOBAL GRADUATE PROGRAM 2014

redefining / standards 

agence c&g - © Photonstop

- Relationships
 - navigability
 - multiplicity
 - dependency
 - aggregation
 - composition

- Generalization / specialization
 - inheritance
 - interfaces

- Keywords
- Notes and Comments

2.3 UML Syntax

As UML diagrams convey precise information there is a precise syntax that should be followed.

Attributes should be shown as: *visibility name: type multiplicity*

Where visibility is one of:-

- '+' public
- '-' private
- '#' protected
- '~' package

and Multiplicity is one of:-

- 'n' exactly n
- '*' zero or more
- 'm..n' between m and n

The following are examples of attributes correctly specified using UML:-

- **custRef: int [1]**

a private attribute custRef is a single int value

this would often be shown as – **custRef: int** However with no multiplicity shown we cannot safely assume a multiplicity of one was intended by the author.

itemCodes: String [1..*]

a protected attribute itemCodes is one or more String values

validCard: boolean

an attribute validCard, of unspecified visibility, has unspecified multiplicity

Operations also have a precise syntax and should be shown as:

visibility name (par1: type1, par2: type2): returntype

where each parameter is shown (in parenthesis) and then the return type is specified.

An example would be

+ addName (newName: String): boolean

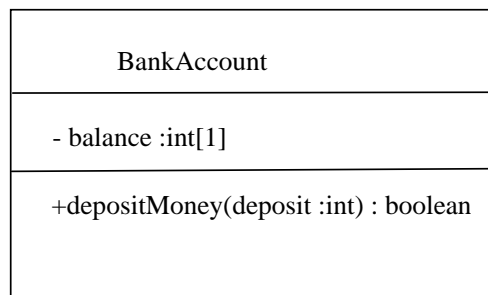
This denotes a public method 'addName' which takes one parameter 'newName' of type String and returns a boolean value.

Activity 2

Draw a diagram to represent a class called 'BankAccount' with a private attribute balance (this being a single integer) and a public method depositMoney() which takes an integer parameter, 'deposit' and returns a boolean value. Fully specify all of this information on a UML class diagram.

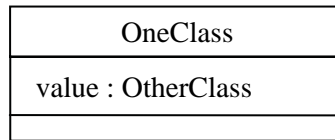
Feedback 2

The diagram below shows this information

**Denoting Relationships**

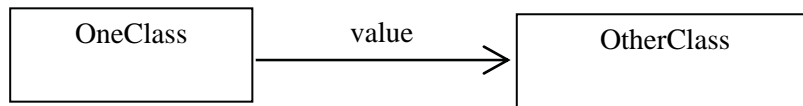
As well as denoting individual classes, Class diagrams denote relationships between classes. One such relationship is called an 'Association'.

In a class attributes will be defined. These could be primitive data types (int, boolean etc.) however attributes can also be complex objects as defined by other classes.



Thus the figure above shows a class ‘OneClass’ that has an attribute ‘value’. This value is not a primitive data type but is an object of type defined by ‘OtherClass’.

We could denote exactly the same information by the diagram below.



We use an association when we want to give two related classes, and their relationship, prominence on a class diagram

The ‘source’ class points to the ‘target’ class.

Empowering People. Improving Business.

BI Norwegian Business School is one of Europe’s largest business schools welcoming more than 20,000 students. Our programmes provide a stimulating and multi-cultural learning environment with an international outlook ultimately providing students with professional skills to meet the increasing needs of businesses.

BI offers four different two-year, full-time Master of Science (MSc) programmes that are taught entirely in English and have been designed to provide professional skills to meet the increasing need of businesses. The MSc programmes provide a stimulating and multi-cultural learning environment to give you the best platform to launch into your career.

- MSc in Business
- MSc in Financial Economics
- MSc in Strategic Marketing Management
- MSc in Leadership and Organisational Psychology

BI NORWEGIAN BUSINESS SCHOOL

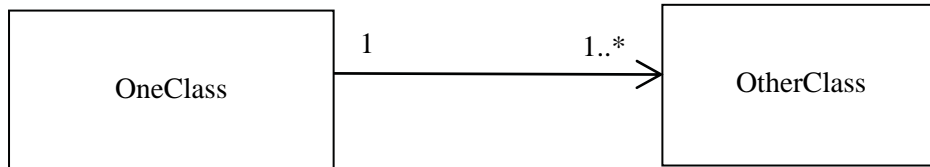
EFMD **EQUIS ACCREDITED**

www.bi.edu/master



Strictly we could use an association when a class we define has a String instance variable – but we would not do this because the String class is part of the Java platform and ‘taken for granted’ like an attribute of a primitive type. This would generally be true of all library classes unless we are drawing the diagram specifically to explain some aspect of the library class for the benefit of someone unfamiliar with its purpose and functionality.

Additionally we can show multiplicity at both ends of an association:



This implies that ‘OneClass’ maintains a collection of objects of type ‘OtherClass’. Collections are an important part of the Java framework that we will look at the use of collections in Chapter 7.

Activity 3

Draw a diagram to represent a class called ‘Catalogue’ and a class called ‘ItemForSale’ as defined below:-

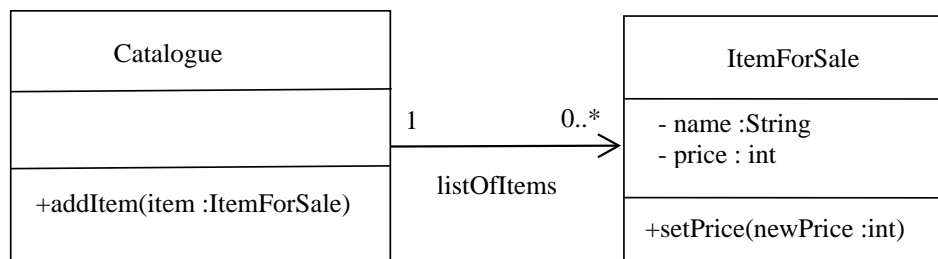
ItemForSale has an attribute ‘name’ of type string and an attribute ‘price’ of type int. It also has a method setPrice() which takes an integer parameter ‘newPrice’.

‘Catalogue’ has an attribute ‘listOfItems’ i.e. the items currently held in the catalogue. As zero or more items can be stored in the catalogue ‘listOfItems’ will need to be an array or collection. ‘Catalogue’ also has one method addItem() which takes an ‘item’ as a parameter (of type ItemForSale) and adds this item to the ‘listOfItems’.

Draw this on a class diagram showing appropriate visibility modifiers for attributes and methods.

Feedback 3

The diagram below shows this information

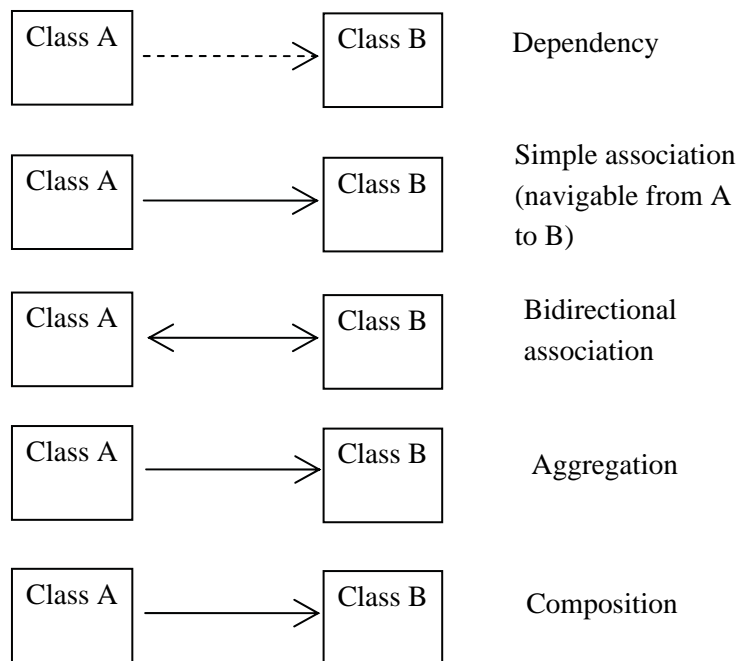


Note: All class names begin in uppercase, attribute and method names begin in lowercase. Also note that the class ItemForSale describes a single item (not multiple items). 'listOfItems' however maintains a list of zero or more individual objects.

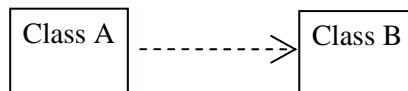
Types of Association

There are various different types of association denoted by different arrows:-

- Dependency,
- Simple association
- Bidirectional association
- Aggregation and
- Composition



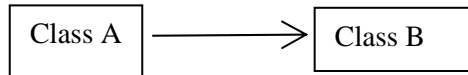
Dependency



- Dependency is the most unspecific relationship between classes (not strictly an 'association')
- Class A in some way uses facilities defined by Class B
- Changes to Class B may affect Class A

Typical use of dependency lines would be where Class A has a method which is passed a parameter object of Class B, or uses a local variable of that class, or calls 'static' methods in Class B.

Simple Association



- In an association Class A 'uses' objects of Class B
- Typically Class A has an attribute of Class B
- Navigability is from A to B:
 - i.e. A Class A object can access the Class B object(s) with which it is associated. The reverse is not true – the Class B object doesn't 'know about' the Class A object

A simple association typically corresponds to an instance variable in Class A of the target class B type.

Example: the **Catalogue** above needs access to 0 or more **ItemsForSale** so items can be added or removed from a **Catalogue**. An **ItemForSale** does not need to access a **Catalogue** in order to set its price or perform some other method associated with the item itself.

Need help with your dissertation?

Get in-depth feedback & advice from experts in your topic area. Find out what you can do to improve the quality of your dissertation!

Get Help Now

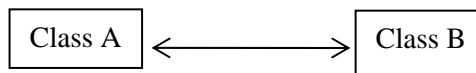


Go to www.helpmyassignment.co.uk for more info



Helpmyassignment

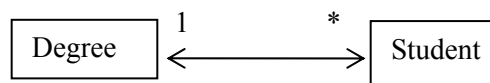


Bidirectional Association

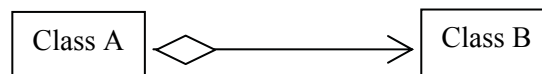
- Bidirectional Association is when Classes A and B have a two-way association
- Each refers to the other class
- Navigability A to B and B to A:
 - A Class A object can access the Class B object(s) with which it is associated
 - Object(s) of Class B 'belong to' Class A
 - Implies reference from A to B
 - Also, a Class B object can access the Class A object(s) with which it is associated

A bidirectional association is complicated because each object must have a reference to the other object(s) and generally bidirectional associations are much less common than unidirectional ones.

An example of a bidirectional association may be between a 'Degree' and 'Student'. ie. given a Degree we may wish to know which Students are studying on that Degree. Alternatively starting with a student we may wish to know the Degree they are studying.

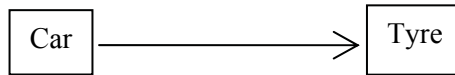


As many students study the same Degree at the same time, but students usually only take one Degree there is still a one to many relationship here.

Aggregation

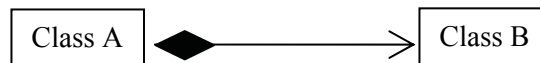
- Aggregation denotes a situation where Object(s) of Class B 'belong to' Class A
- Implies reference from A to B
- While aggregation implies that objects of Class B belong to objects of Class A it also implies that object of Class B retain an existence independent of Class A. Some designers believe there is no real distinction between aggregation and simple association

An example of aggregation would be between a Class Car and a Class Tyre



We think of the tyres as belonging to the car they are on, but at the garage they may be removed and placed on a rack to be repaired. Their existence isn't dependent on the existence of a car with which they are associated.

Composition



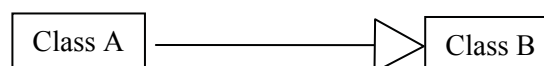
- Composition is similar to aggregation but implies a much stronger belonging relationship i.e. Object(s) of Class B are 'part of' a Class A object
- Again implies reference from A to B
- Much 'stronger' than aggregation in this case Class B objects are an integral part of Class A and in general objects of Class B never exist other than as part of Class A, i.e. they have the same 'lifetime'

An example of composition would be between Points, Lines and Shapes as elements of a Picture. These objects can only exist as part of a picture, and if the picture is deleted they are also deleted.

As well as denoting associations, class diagrams can denote:-

- Inheritance,
- Interfaces,
- Keywords and
- Notes

Inheritance

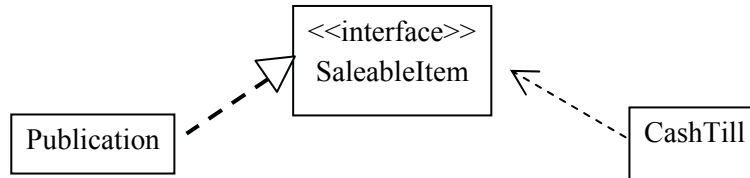


- Aside from associations, the other main modelling relationship is inheritance:
- Class A 'inherits' both the interface and implementation of Class B, though it may override implementation details and supplement both.

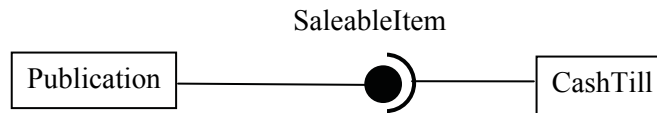
We will look at inheritance in detail in Chapter 3.

Interfaces

- Interfaces are similar to inheritance however with interfaces only the interface is inherited. The methods defined by the interface must be implemented in every class that implements the interface.
- Interfaces can be represented using the <<interface>> keyword:



There is also a shorthand for this



In both cases these examples denote that the SaleableItem interface is **required by** CashTill and **implemented by** Publication.

Brain power



Plug into The Power of Knowledge Engineering.
Visit us at www.skf.com/knowledge

By 2020, wind could provide one-tenth of our planet's electricity needs. Already today, SKF's innovative know-how is crucial to running a large proportion of the world's wind turbines.

Up to 25 % of the generating costs relate to maintenance. These can be reduced dramatically thanks to our systems for on-line condition monitoring and automatic lubrication. We help make it more economical to create cleaner, cheaper energy out of thin air.

By sharing our experience, expertise, and creativity, industries can boost performance beyond expectations. Therefore we need the best employees who can meet this challenge!

The Power of Knowledge Engineering





NB the dotted-line version of the inheritance line/arrow which shows that Publication ‘implements’ or ‘realizes’ the SaleableItem interface.

The “ball and socket” notation is new in UML 2 – it is a good visual way of representing how interfaces connect classes together.

We will look at the application of interfaces in more detail in Chapter 4.

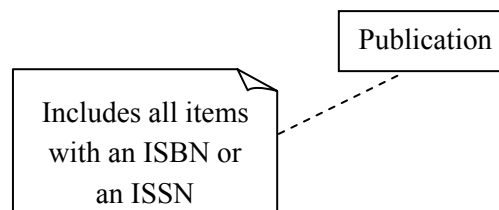
Keywords

UML defines keywords to refine the meaning of the graphical symbols

We have seen <<interface>> and we will also make use of <<abstract>> but there are many more.

An abstract class may alternatively be denoted by showing its name in *italics* though this is perhaps less obvious to a casual reader.

Notes



Finally we can add notes to comment on a diagram element. This gives us a ‘catch all’ facility for adding information not conveyed by the graphical notation.

Activity 4

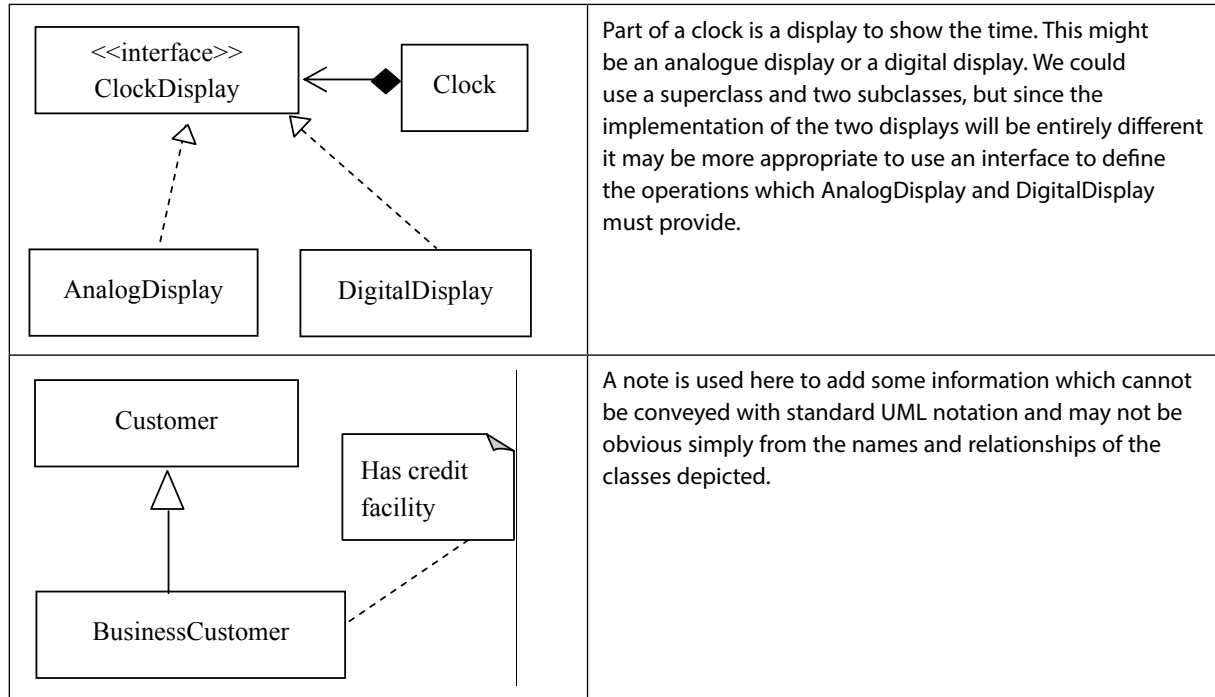
From your own experience, try to develop a model which illustrates the use of the following elements of UML Class Diagram notation:

- simple association
- bidirectional association
- aggregation (tricky!)
- composition
- association multiplicity
- generalization (inheritance)
- interfaces
- notes

For this exercise concentrate on the relationships between classes rather than the details of their members. Explain and discuss your model with other students and your tutor.

To help you get started some small examples are given below.

<pre> classDiagram Student "1" -- "0..4" Transcript </pre>	<p>In a University administration system we might produce a transcript of results for each year the student has studied (including a possible placement year).</p> <p>This association relationship is naturally unidirectional – given a student we might want to find their transcript(s), but it seems unlikely that we would have a transcript and need to find the student to whom it belonged.</p>
<pre> classDiagram Reader "0..1" <--> "0..8" BorrowedBook </pre>	<p>In a library a reader can borrow up to eight books. A particular book can be borrowed by at most one reader.</p> <p>We might want a bidirectional relationship as shown here because, in addition to being able to identify all the books which a particular reader has borrowed, we might want to find the reader who has borrowed a particular book (for example to recall it in the event of a reservation).</p>
<pre> classDiagram Body "1" *-- "2" Hand Hand "1" *-- "1" Thumb Hand "1" *-- "4" Finger </pre>	<p>This might be part of the model for some kind of educational virtual anatomy program.</p> <p>Composition – the “strong” relationship which shows that one object is (and has to be) part of another seems appropriate here.</p> <p>The multiplicities would not always work for real people though – they might have lost a finger due to accident or disease, or have an extra one because of a genetic anomaly.</p> <p>And what if we were modelling the “materials” in a medical school anatomy lab? A hand might not always be part of a body! Perhaps the “weaker” aggregation relationship would reflect this better.</p>
<pre> classDiagram class BankAccount { <<abstract>> } class CurrentAccount class SavingsAccount class Customer BankAccount < -- CurrentAccount BankAccount < -- SavingsAccount BankAccount "*" -- "1..2" Customer </pre>	<p>A customer can have any number of bank accounts, and a bank account can be held by one person or two people (a “joint account”). We have suppressed the navigability of this relationship, perhaps because we have not yet decided this issue.</p> <p>A bank account must either be a current account or a savings account – hence BankAccount itself is abstract.</p> <p>(We could have shown this using italics rather than the <<abstract>> keyword)</p> <div style="border: 1px solid black; padding: 5px; margin: 10px auto; width: fit-content;"> <i>Bank Account</i> </div>



Feedback 4
 There is no specific feedback for this activity.

“I studied English for 16 years but...
 ...I finally learned to speak it in just six lessons”

Jane, Chinese architect

ENGLISH OUT THERE

Click to hear me talking before and after my unique course download



2.4 UML Package Diagrams

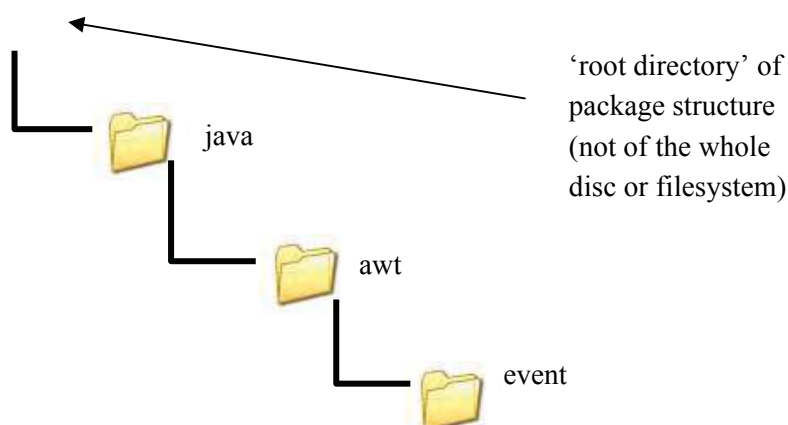
While class diagrams are the most commonly used diagram of those defined in UML notation, and we will make significant use of these throughout this book, there are other diagrams that denote different types of information. Here we will touch upon three of these:-

- Package Diagrams
- Object Diagrams and
- Sequence Diagrams

World maps, country maps and city maps all show spatial information, just on different scales and with differing levels of detail. Large OO systems can be made up of hundreds, or potentially thousands, of classes and thus if the class diagram was the only way to represent the architecture of a large system it would become overly large and complex. Thus, just as we need world maps, we need package diagrams to show the general architecture of a large system. Even modest systems can be broken down into a few basic components i.e. packages. We will see an example of packages in use in Chapter 11. For now we will just look at the package diagramming notation.

A package is not just a visual representation of a group of classes instead a 'package' is a directory containing a group of related classes (and interfaces). Packages allow us to provide a level of organisation and encapsulation above that of individual classes and all of the standard Java platform classes are arranged in a single large package hierarchy. Similarly we can also arrange our own classes using the Java package mechanism.

Packages are described as a series of dot-separated names, e.g. `java.awt.event`. The names correspond to a series of sub-directories in the file system, e.g.

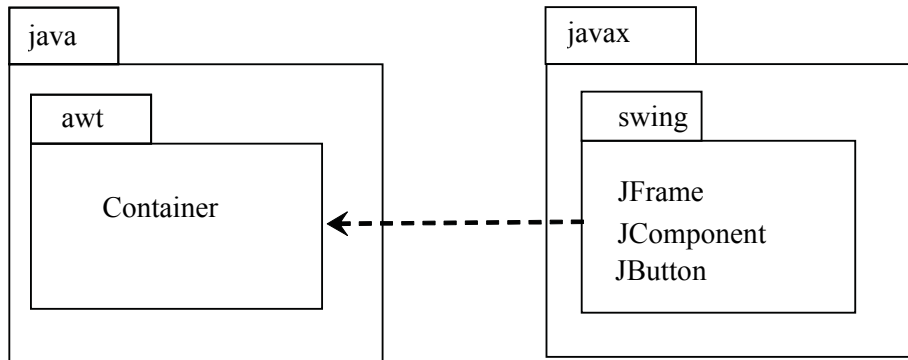


A large Java development should be split into suitable packages at the design stage UML provides a 'Package Diagram' to represent the relationships between classes and packages.

We can depict

- classes within packages
- nesting of packages
- dependencies between packages

In the diagram below we see two packages:- 'java' and 'javax'

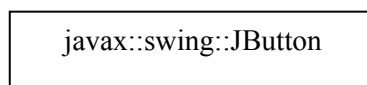


Looking at this more closely we can see that inside the 'java' package is another called 'awt' and inside 'javax' is a package called 'swing'.

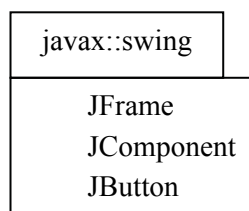
The package 'awt' contains a class 'Container' and 'javax' contains three classes 'JFrame', 'JComponent' and 'JButton'. Finally we show that the javax.swing package has dependencies on the java.awt package.

Note that the normal UML principle of suppression applies here – both java.awt and javax.swing contain many more classes, and 'java' contains other sub-packages, but we simply choose not to show them.

In the diagram below we have an alternative way of indicating that a JButton is in the javax.swing package.



And again below a form which shows all three classes more concisely than at the top.



These different representations will be useful in different circumstances depending on what a package diagram is aiming to convey.

Package Naming

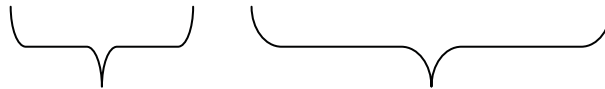
By convention, package names are normally in lowercase

For local individual projects packages could be named according to personal preference, e.g.

```
mssystem
mssystem.interface
mssystem.engine
mssystem.engine.util
mssystem.database
```

However, packages are often distributed and to enable this packages need globally unique names, thus a naming convention has been adopted based on URLs

uk.co.ebay.www.department.project.package



What do you want to do?

No matter what you want out of your future career, an employer with a broad range of operations in a load of countries will always be the ticket. Working within the Volvo Group means more than 100,000 friends and colleagues in more than 185 countries all over the world. We offer graduates great career opportunities – check out the Career section at our web site www.volvogroup.com. We look forward to getting to know you!

VOLVO
AB Volvo (publ)
www.volvogroup.com

VOLVO TRUCKS | RENAULT TRUCKS | MACK TRUCKS | VOLVO BUSES | VOLVO CONSTRUCTION EQUIPMENT | VOLVO PENTA | VOLVO AERO | VOLVO IT
VOLVO FINANCIAL SERVICES | VOLVO 3P | VOLVO POWERTRAIN | VOLVO PARTS | VOLVO TECHNOLOGY | VOLVO LOGISTICS | BUSINESS AREA ASIA

Download free eBooks at bookboon.com



Click on the ad to read more

Part based on organisation URL (e.g. ww.ebay.co.uk) reversed, though this does **not** specifically imply you can download the code there.

Part distinguishing the particular project and component or subsystem which this package contains.

Note on a package diagram each element is not separated by a '?' but by '::':

Activity 5

You and a flatmate decide to go shopping together. For speed split the following shopping list into two halves – items to be collected by you and items to be collected by your flatmate.

Apples, Furniture polish, Pears, Carrots, Toilet Rolls, Potatoes, Floor cleaner.
Matches, Grapes

Feedback 5

To make your shopping efficient you probably organised your list into two lists of items that are located in the same parts of the shop:-

List 1	List 2
Apples,	Furniture polish,
Pears,	Floor cleaner
Grapes	Matches
Carrots,	Toilet Rolls,
Potatoes	

Activity 6

You run a team of three programmers and are required to write a program in Java to monitor and control a network system. The system will be made up of seven classes as described below. Organise these classes into three packages. Each programmer will then be responsible for the code in one package. Give the packages any name you feel appropriate.

Main	this class starts the system
Monitor	this class monitors the network for performance and breaches in security
Interface	this is a visual interface for entire system
Reconfigure	this allows the network to be reconfigured
RecordStats	this stores data regarding the network in a database
RemoteControl	this allows some remote control over the system via telephone
PrintReports	this uses the data stored in the database to print management reports for the organisations management.

Feedback 6

When organising a project into packages there is not always 'one correct answer' but if you organise your classes into appropriate packages (with classes that have related functionality) you improve the encapsulation of the system and improve the efficiency of your programmers. A suggested solution to activity 6 is given below.

```
interface
    Main
    Interface
    RemoteControl

network
    Monitor
    Reconfigure

database
    RecordStats
    PrintReports
```

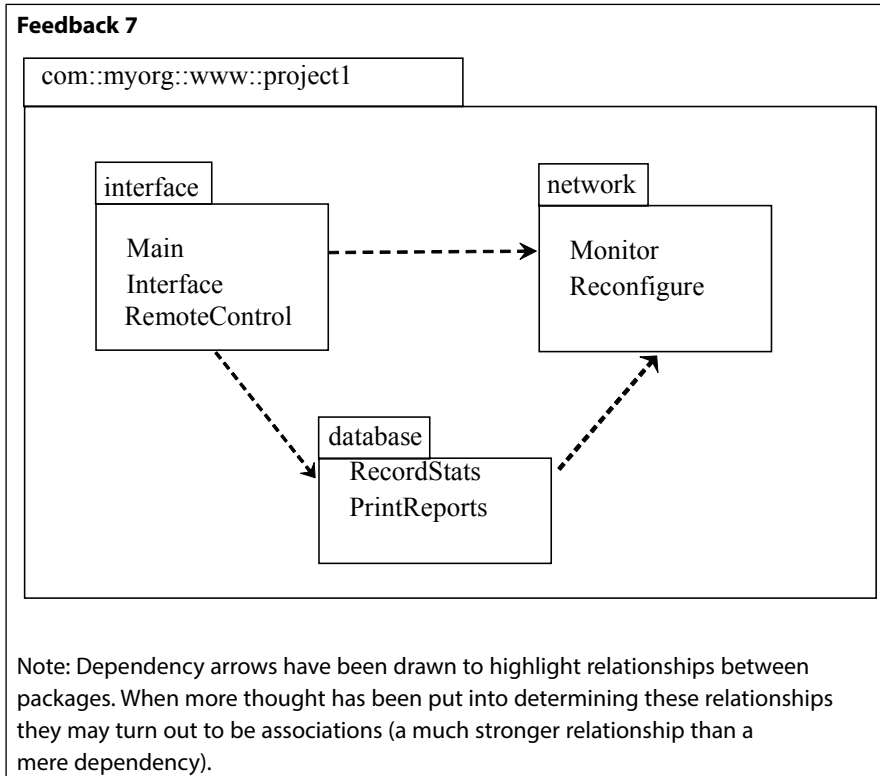
Activity 7

Assume the URL of your organisation is 'www.myorg.com' and the three packages and seven classes shown below are all part of 'project1'. Draw a package diagram to convey this information.

```
interface
    Main
    Interface
    RemoteControl

network
    Monitor
    Reconfigure

database
    RecordStats
    PrintReports
```



2.5 UML Object Diagrams

Class diagrams and package diagrams allow us to visualise and discuss the architecture of a system however at times we wish to discuss the data a system processes. Object diagrams allow us to visual one instance of time and the data that a system may contain in that moment.

Object diagrams look superficially similar to class diagrams however the boxes represent specific instances of objects.

Boxes are titled with:-

objectName: ClassName

As each box describes a particular object at a specific moment in time the box contains attributes and their values (at that moment in time).

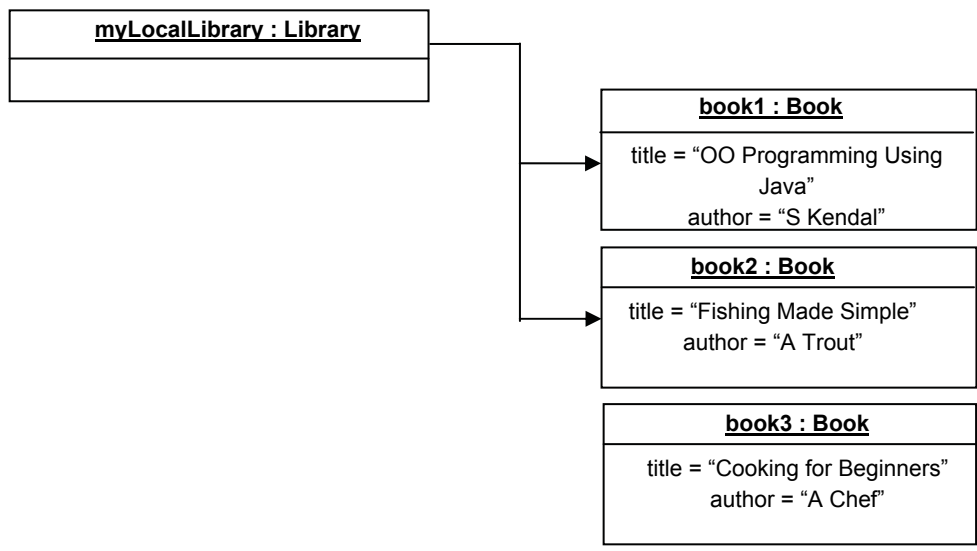
attribute = value

These diagrams are useful for illustrating particular ‘snapshot’ scenarios during design.

The object diagram below shows several object that may exist at a moment in time for a library catalogue system. The system contains two classes:-

Book, which store the details of a book and

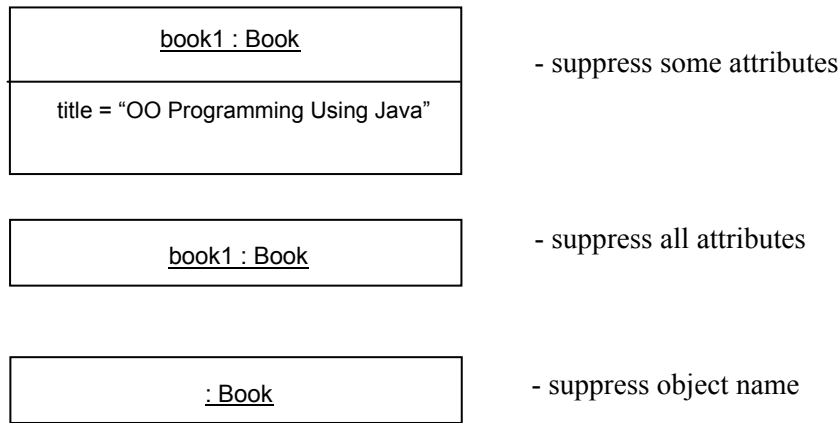
Library, which maintains a collection of books. With books being added, searched for or removed as required.



Looking at this diagram we can see that at a particular moment in time, while three books have been created only two have been added to the library. Thus if we were to search the library for ‘Cooking for Beginners’ we would not expect the book to be found.

As with class diagrams, elements can be freely suppressed on object diagrams.

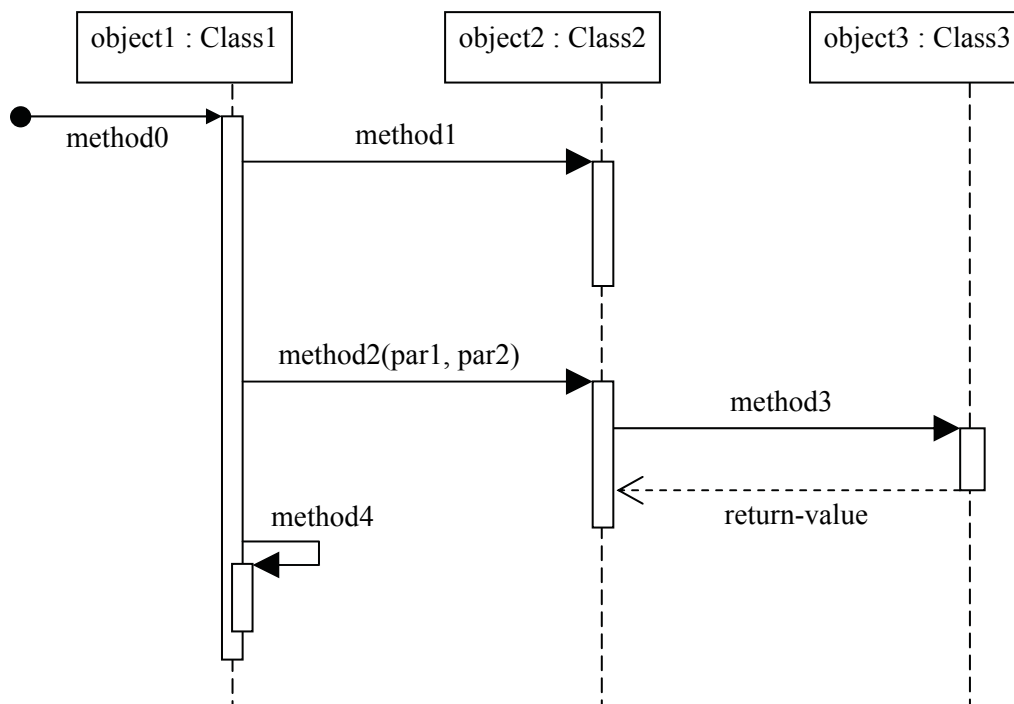
For example, all of these are legal:



2.6 UML Sequence Diagrams

Sequence diagrams are entirely different from class diagrams or object diagrams. Class diagrams describe the architecture of a system and object diagrams describe the state of a system at one moment in time. However sequence diagrams describe how the system works over a period of time. Sequence diagrams are 'dynamic' rather than 'static' representations of the system. They show the sequence of method invocations within and between objects over a period of time. They are useful for understanding how objects collaborate in a particular scenario.

See the example below:-



We have three objects in this scenario. Time runs from top to bottom, and the vertical dashed lines (lifelines) indicate the objects' continued existence through time.

This diagram shows the following actions taking place:-

- Firstly a method call (often referred to in OO terminology as a message) to method0() comes to object1 from somewhere – this could be another class outside the diagram.
- object1 begins executing its method0() (as indicated by the vertical bar (called an activation bar) which starts at this point.
- object1.method0() invokes object2.method1() – the activation bar indicates that this executes for a period then returns control to method0()
- Subsequently object1.method0() invokes object2.method2() passing two parameters
- method2() subsequently invokes object3.method3(). When method3() ends it passes a return value back to method2()
- method2() completes and returns control to object1.method0()
- Finally method0() calls another method of the same object, method4()

Selection and Iteration

The logic of a scenario often depends on selection ('if') and iteration (loops).

There is a notation ('interaction frames') which allow ifs and loops to be represented in sequence diagrams however these tend to make the diagrams cluttered.

Sequence diagrams are generally best used for illustrating particular cases, with the full refinement reserved for the implementation code.

Fowler ("UML Distilled", 3rd Edn.) gives a brief treatment of these constructs.

2.7 Summary

UML is not a methodology but a precise diagramming notation.

Class diagrams and package diagrams are good for describing the architecture of a system. Object diagrams describe the data within an application at one moment in time and sequence diagrams describe how a system works over a period of time.

UML gives different meaning to different arrows therefore one must be careful to use the notation precisely as specified.

With any UML diagram suppression is encouraged – thus the author of a diagram can suppress any details they wish in order to convey essential information to the reader.



Potential for exploration

Potential for development

ENGINEERS, UNIVERSITY GRADUATES & SALES PROFESSIONALS
Junior and experienced F/M

Total will hire 10,000 people in 2014. Why not you?

Are you looking for work in process, electrical or other types of engineering, R&D, sales & marketing or support professions such as information technology?

We're interested in your skills.

Join an international leader in the oil, gas and chemical industry by applying at

www.careers.total.com
More than 700 job openings are now online!



orc.fr Copyright: Total/Corbis



TOTAL
COMMITTED TO BETTER ENERGY

