CITY UNIVERSITY LONDON

IN2013 Lecture 3

Object-oriented Analysis and Design:

Introduction to Object Orientation, and Class Diagrams

Dr Peter Popov

14th October 2025

1

Map

- Part 1: An Introduction to object-oriented approach to software development
- Part 2: Analysis - finding analysis classes
- Part 3: Relationships between objects and classes
- Part 4: Inheritance and polymorphism
- Part 5: Noun/Verb analysis in practice
  - a worked-out example to be discussed in class.

2

Part 1: An Object-Oriented Approach to Software Development – An Introduction

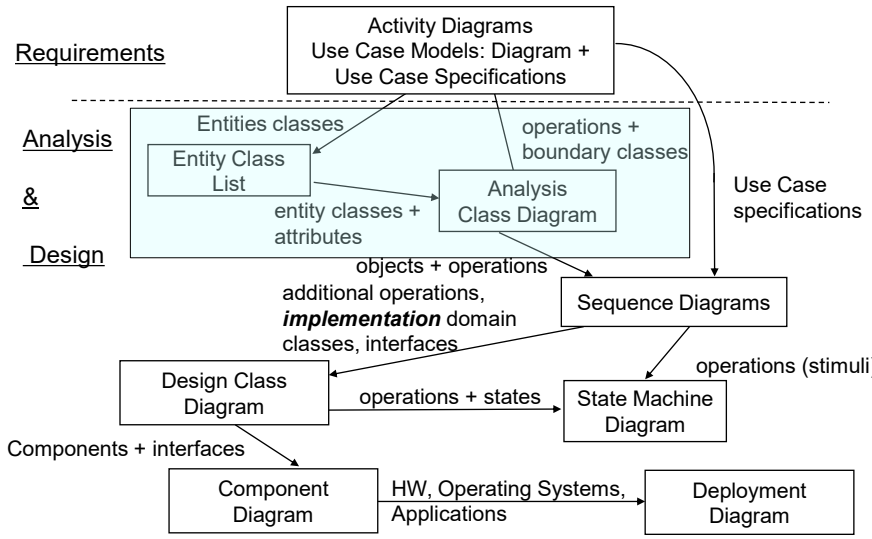
3

Objectives

- Introduce the object-oriented approach to software development
  - What "objects" and "classes" mean in the OO approach
- Introduce the ***analysis models*** and their role in software development
- Show how to specify classes using UML

4

From Requirements to Analysis and Design



5

Analysis

Building a logical model of the system to be developed:

- Analysis classes
  - describing what data it must store and what changes or queries will be required on these data
- Use-case realisations
  - describing the sequences of queries/changes of this information

This set of models:

- helps developers elicit the key abstractions (in the *problem domain*) from the requirements
- gives an initial shape to the structure of the system
- is used as an input to design and implementation

6

Structural Model

A formal way of representing the objects that are used and created by a business system (i.e., in the ***problem domain***)

- People
- Places
- Things

Examples of problem domain: banking, health service, manufacturing, games, etc.

Drawn using an iterative process

- First drawn in a conceptual, business-centric way
- These will be later refined (in design) adding elements from the ***implementation/solutions domain***, e.g., databases, files, communications, graphic user interface, etc..

7

What is Object Orientation?

A modelling and implementation approach for software systems, based on these principles:

- Seeing the world in terms of ***objects***
- Specifying systems as collections of ***interacting objects***
- Protecting data (information hiding and access control) through ***encapsulation***
- Abstract descriptions of *types* of objects: ***classes***
  - to simplify description of systems with many similar objects (describe the brick type you need for your wall, not every single brick)
- and (we'll see later) structuring these descriptions through
  - ***generalisation/specialisation*** of classes
  - uniform access through ***polymorphism***

8

OO vs. non-OO paradigms

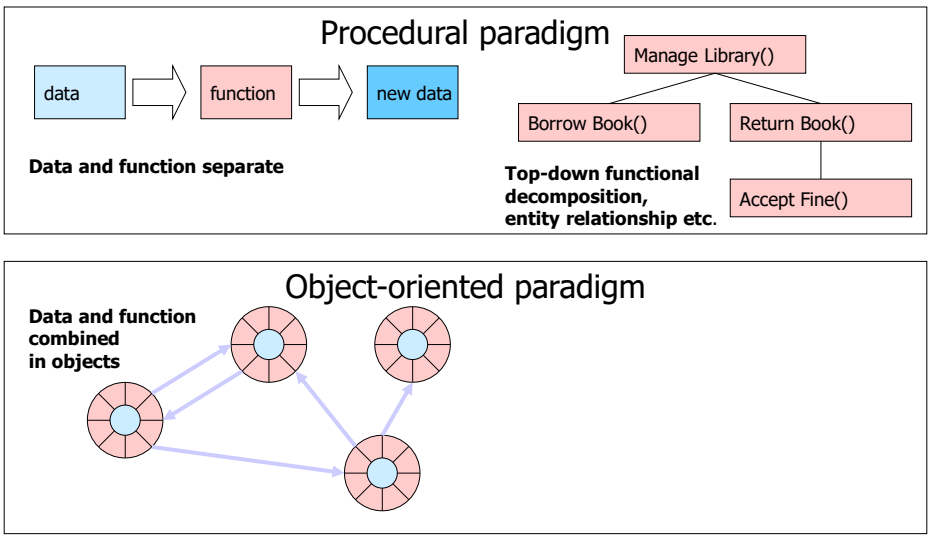
Handling complexity by, "divide and conquer" was used long before object orientation came about:

- process-oriented paradigm
  - according to steps or functions e.g. structured methods:
  - functions (behaviour) and data (information held) are treated ***separately***
  - e.g. SSADM (Cutts 1987), SSA (de Marco 1978), SADT (Ross 1977)
- object-oriented paradigm
  - according to behaviour of autonomous objects
  - data and the functions that use that data are ***encapsulated together***

Both valid, but current claims for superiority of O-O

- stronger framework
- reuse of common abstractions
- resilient under change

Different paradigms…



History of OO

Term "object-oriented" first applied to Smalltalk • *Class* and *inheritance* from simulation languages • *Structural feature* and *functional abstractions* from LISP	"Object-oriented" user interfaces • *WIMP* (OO interfaces) • OO required to manage complexity	OO analysis and design • Benefits of OO • Reuse
invention	**confusion**	**ripening**
1970	1980	1990
	Artificial Intelligence • *Frames* and *actors*	OO extended into: • Databases • Standards

Some Definitions of Objects

An object is:

"An abstraction of something in a problem domain, reflecting the capabilities of a system to keep information about it, interact with it, or both"

(Coad & Yourdon 1991)

"An entity able to save a state (information) and which offers a number of operations (behaviour) to either examine or affect this state"

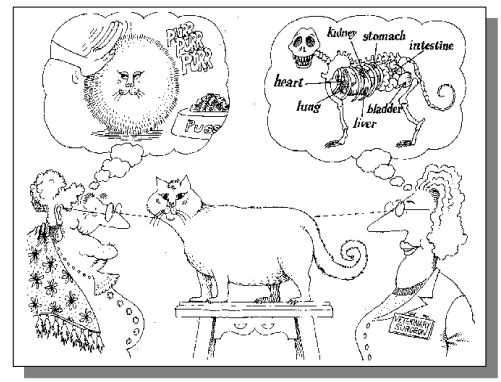
(Jacobson 1992)

Who is Peter Coad (https://en.wikipedia.org/wiki/Peter_Coad)
Who is Ivar Jacobson (https://en.wikipedia.org/wiki/Ivar_Jacobson)

Objects & Abstraction

"An abstraction denotes the ***essential*** characteristics of an object that distinguish it from all other kinds of objects and thus provides **crisply defined conceptual boundaries**, relative to the ***perspective of the viewer***"

- (Grady Booch 1994)



Who is Grady Booch (https://en.wikipedia.org/wiki/Grady_Booch)

What are objects?

Objects consist of data and functions packaged together in a reusable unit. Objects *encapsulate* data

Every object is an instance of some *class* which defines the common set of *features* (attributes and operations) shared by all of its instances. Objects have:

- Attribute values – the data part
- Operations – the behaviour part

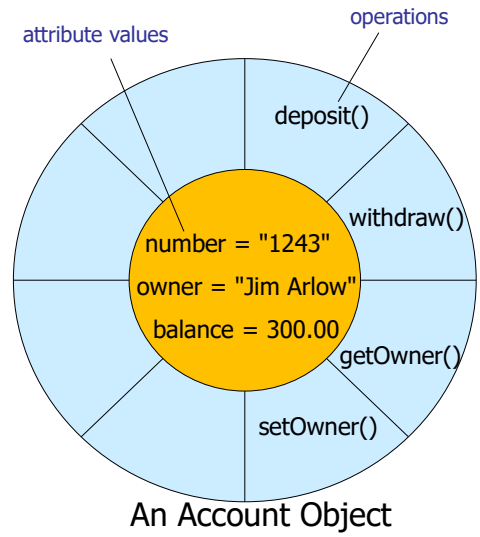
All objects have:

- *Identity*: Each object has its own unique ***identity*** and can be accessed by a unique handle (e.g. the location in memory the instance is stored)
- *State*: This is the actual ***data values*** stored in an object at any point in time
- *Behaviour*: The set of ***operations*** that an object can perform

Encapsulation

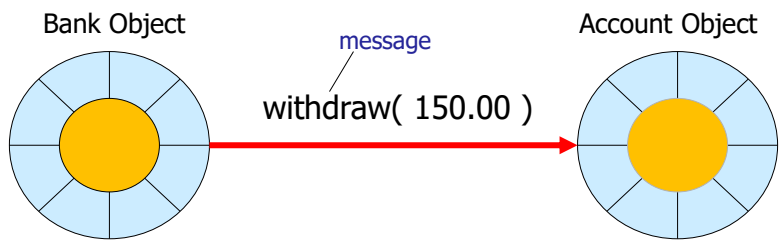
Data is hidden inside the object. The only way to access the data is via one of the operations

This is *encapsulation* or *data hiding* and it is a very powerful idea. It leads to more robust software and reusable code.



Messaging

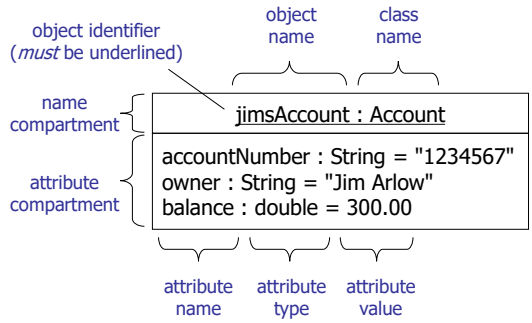
In OO systems, objects send messages to each other over **links**
These messages cause an object to invoke an operation

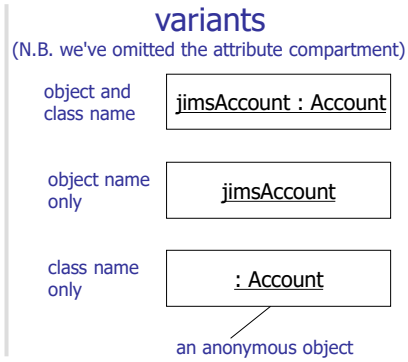


the Bank object sends the message "withdraw 150.00" to an Account object.

the Account object responds by invoking its withdraw operation. This operation decrements the account balance by 150.00.

UML Object Syntax





All objects of a particular class have the same set of operations. They are not shown on the object diagram, they are shown on the class diagram (see later)

Attribute types are often omitted to simplify the diagram

Naming:

- object and attribute names in lowerCamelCase
- class names in UpperCamelCase

17

What are classes?

Every object is an instance of **one** class - the class describes the "type" of the object

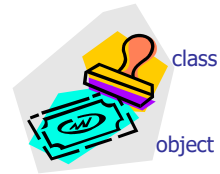
Classes allow us to model sets of objects that have the *same* set of features - a class acts as a template for objects:

- The class determines the structure (set of features) of all objects of that class
- All objects of a class *must* have the same set of operations, *must* have the same attributes, but *may* have **different attribute values**

Classification is one of the most important ways we have of organising our view of the world

Think of classes as being like:

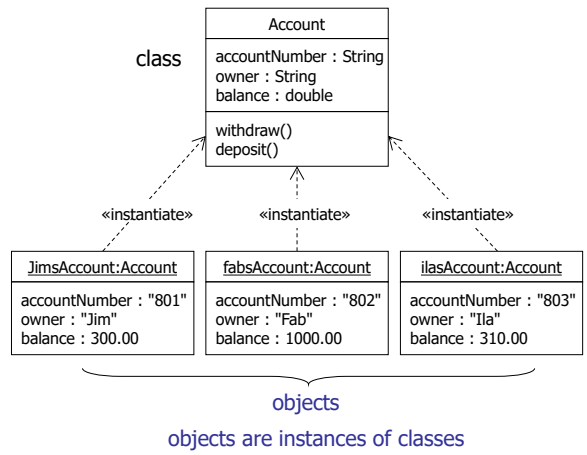
- Rubber stamps
- Cookie cutters



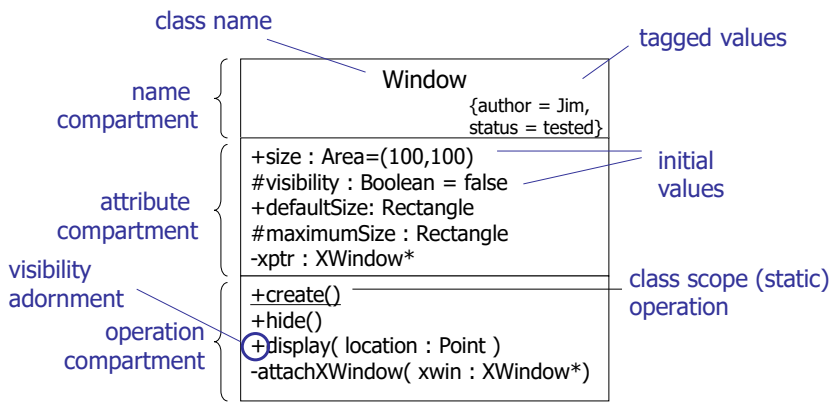
18

Classes and objects

- Objects are instances of classes
- Instantiation is the creation of new instances of model elements
- Most classes provide special operations called *constructors* to create instances of that class. These operations have *class-scope* i.e. they belong to the class itself rather than to objects of the class



UML class notation



Classes are named in UpperCamelCase
Use descriptive names that are nouns or noun phrases
Avoid abbreviations!

Takeaway Messages (Part 1)

- In object-oriented approach to software development implements the required functionality in the form of ***interacting objects***
- Objects encapsulate data (attributes) and operations.
  - Data is hidden and can only be accessed via the operations defined for the objects
  - Objects interact with one another via messages
  - Messages can only be passed if a link exists between the objects
- Objects are instances of Classes
  - Many objects can be derived from the same class.
  - Objects are created by calling the constructor of a class (a special operation)

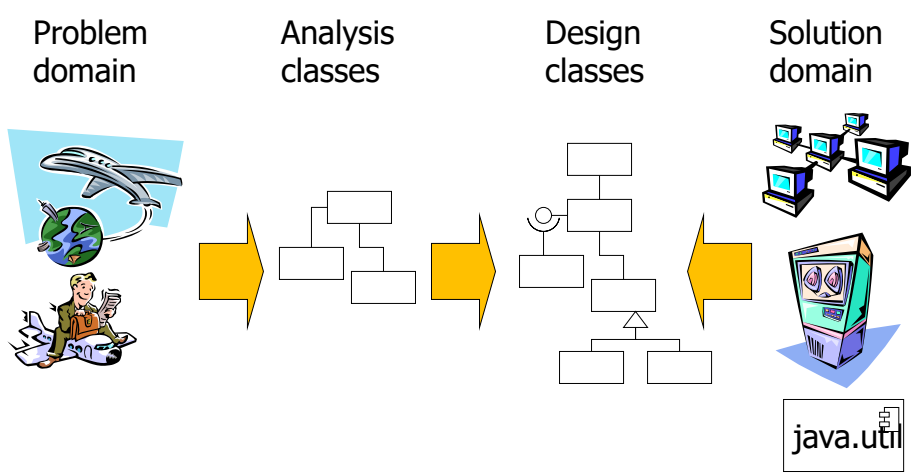
Further Reading (Part 1)

Jim Arlow's book "UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design", Edition 2: Chapter 7.

Part 2: Analysis - finding analysis classes

23

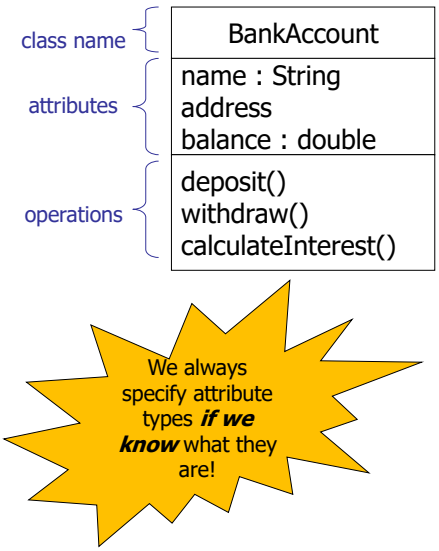
OO analysis and design



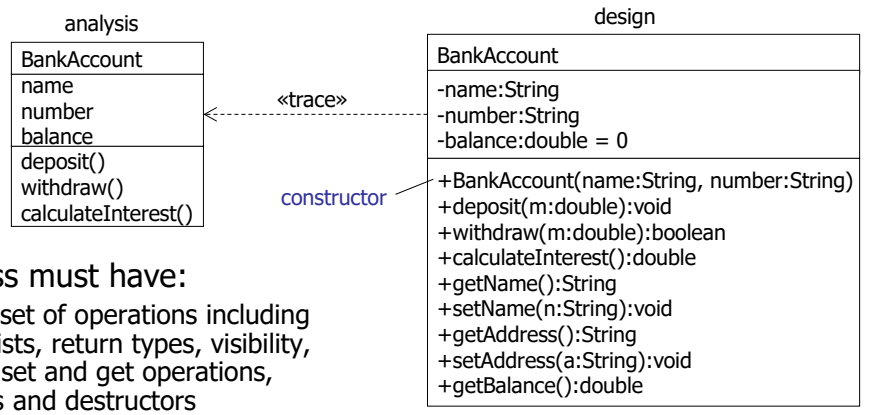
24

What are Analysis classes?

- Analysis classes represent a crisp abstraction in the ***problem domain***
  - They may ultimately be refined into one or more design classes
- All classes in the *Analysis model* should be Analysis classes
- Analysis classes have:
  - A very "high level" set of attributes. They *indicate* the attributes that the design classes *might* have.
  - Operations that specify at a high level the ***key services*** that the class must offer. In Design, they will become actual, implementable, operations.
- Analysis classes must map onto ***real-world business concepts***



Analysis vs. Design classes



A design class must have:

- A complete set of operations including parameter lists, return types, visibility, exceptions, set and get operations, constructors and destructors
- A complete set of attributes including types and default values

What makes a good analysis class?

Its name reflects its intent

It is a *crisp abstraction* that models one specific element of the problem domain
- It maps onto a clearly identifiable feature of the problem domain

It has *high cohesion*
- Cohesion is the degree to which a class models a ***single*** abstraction
- Cohesion is the degree to which the *responsibilities* of the class are semantically related
  - For instance, a BankAccount should allow one to deposit and withdraw from the funds. BankAccountHolder (name, address, etc.) is a different abstraction and should be handled by a different class.

It has *low coupling*
- Coupling is the degree to which one class depends on others

Rules of thumb:
- 3 to 5 responsibilities per class
- Each class collaborates with others
- Beware many very small classes
- Beware few but very large classes
- Beware of "functoids"
- Beware of "omnipotent" classes
- Avoid deep inheritance trees

A *responsibility* is a contract or obligation of a class - it resolves into operations and attributes

Finding classes

Perform noun/verb analysis on documents:
- Nouns are candidate classes or attributes
- Verbs are candidate *responsibilities/operations* or *associations*

Beware of spurious classes:
- Look for *synonyms* - different words that mean the same
- Look for *homonyms* - the same word meaning different things

Look for "hidden" classes!
- Classes that don't appear as nouns in the texts (use case specifications, initial statement of requirements, etc.)

Other methods exist such as CRC cards, which we will cover in the next lecture.

Use patterns – often domain neutral and reusable.

Noun/verb analysis procedure

Make a list of nouns and noun phrases
- These are candidate ***classes*** or ***attributes***
- Each noun/noun phrase can be either:
  - A class
  - An attribute
  - Neither, in which case it is *ignored* (because it is neither a class nor an attribute). Often, a noun may refer to *an actor*.
    - Actors are *not modelled* in class diagram.
    - But an actor may lead to a class (e.g. User leads to a UserAccount) held in the Analysis model.
      - All actors must interact with the system via ***boundary classes*** (introduced in the next lecture)

Make a list of verbs and verb phrases
- These are candidate ***operations*** or ***named associations***
- Analysis consists of passing judgement whether a verb phrase is:
  - An operation
  - A named association
  - Ignored in analysis

Tentatively assign attributes and operations to classes

29

Other sources of classes

- Physical objects
- Paperwork, forms etc.
  - Be careful with this one – if the existing business process is poor, then the paperwork that supports it might be irrelevant.
  - But be aware not to try to automate "archaic" paper-based processes.
    - Business processes may require redesign before automation ("Principle of Convergent Engineering", the term was coined by David Taylor in 1995).
- Known interfaces to the outside world
- Conceptual entities that form a cohesive abstraction e.g. LoyaltyProgramme.

30

Example - food manufacturing company

Tangible objects:
- one packet of herbal tea
- invoice 63501 sent to A Farm, Lincolnshire

Role objects:
- plant operator
- quality controller

Interaction objects:
- a purchasing transaction
- order for new stock

Event objects:
- processing plant breaks down

Organisational objects:
- Marketing Department

Takeaway Messages (Part 2)

- Noun verb analysis is used to identify classes from the problem domain
- This is a systematic (almost mechanistic) process of analysing existing text.
  - A noun can be:
    - A class from the problem domain
    - An attribute of a class
    - Neither, hence should be ignored and not included in the class diagram. *Actors* are often ignored (as they are outside the system) unless the system stored details about them (username/password/role)
  - A verb can be:
    - An operation of a class
    - An association between two classes
    - Neither (e.g. a generic verb) and should be ignored.
- Noun/verb analysis is a "best effort activity" and ***does not guarantee*** that all important classes will be captured but is a good starting point for analysis.

Further Reading (Part 2)

Jim Arlow's book "UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design", Edition 2: Chapter 8.

33

Part 3: Relationships between objects and classes

34

What is a relationship?

- A *relationship* is a connection between modelling elements
- In this section we look at:
  - *Links* between objects
  - *Associations* between classes
    - Association classes
    - Aggregation
    - Composition

These are essential in *design*, not in analysis. Will explain these types of associations between classes later in this module.

What is a link?

Links are connections *between objects*

- Think of a link as a telephone line connecting you and a friend. You can send messages (i.e. make phone calls) back and forth using this link.

Links are the way that objects ***communicate.***

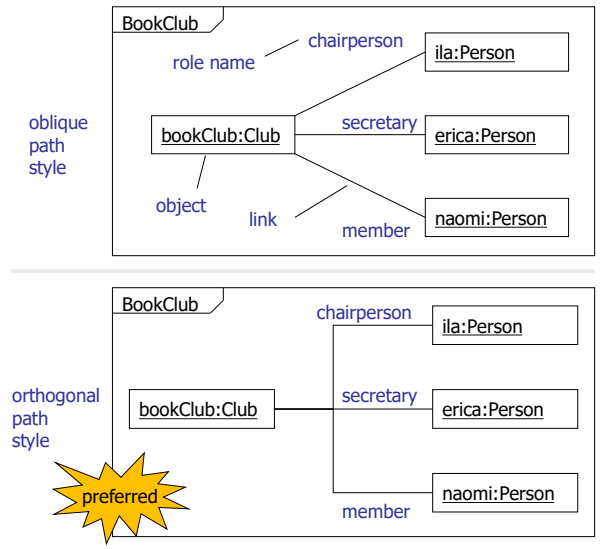
- Objects send messages to each other via links. Only objects with links can communicate.
- Messages invoke operations.

OO programming languages implement links as ***object references or pointers***. These are unique handles that refer to specific objects

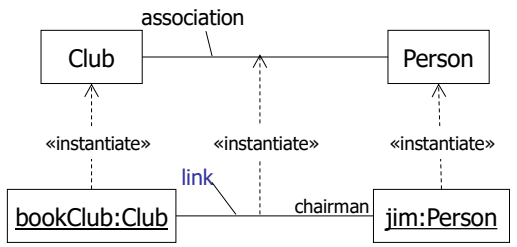
- When an object has a reference to another object, we say that there is a *link* between the objects.

Object diagrams

- Paths in UML diagrams (lines to you and me!) can be drawn as orthogonal, oblique or curved lines
- We can combine paths into a tree *if* each path has the same properties

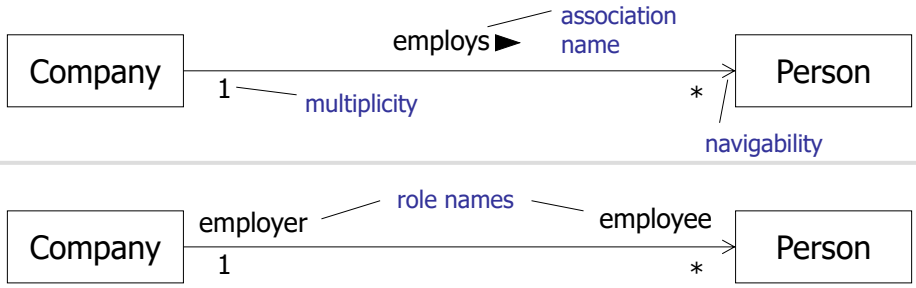


What is an association?



- Associations are relationships between classes.
- Associations between classes *indicate that there are links between objects* of those classes.
- A link is an instantiation of an association just as an object is an instantiation of a class.

Association syntax

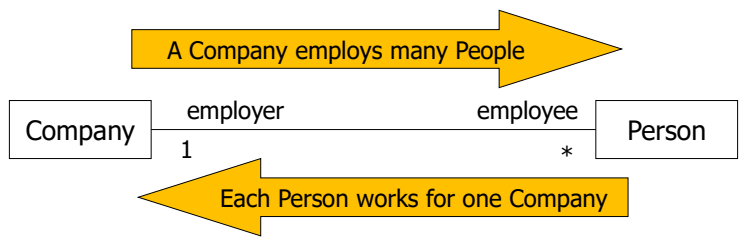


An association can have role names *or* an association name
- It's bad style to have both!

The black triangle indicates the ***direction*** in which the association name is read:
- "A Company employs many Persons"

39

Multiplicity



Multiplicity is a constraint that specifies the ***number of objects*** that can participate in a relationship at *any point in time*

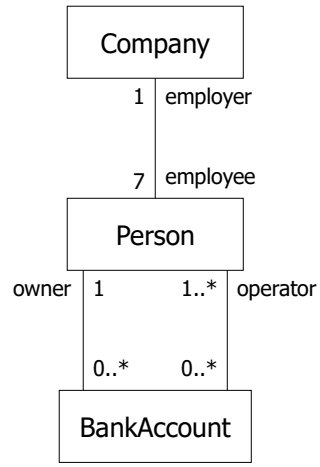
If multiplicity is not explicitly stated in the model, then it is ***undecided*** – *there is no default multiplicity*

multiplicity syntax: minimum..maximum	
0..1	zero or 1
1	exactly 1
0..*	zero or more
*	zero or more
1..*	1 or more
1..6	1 to 6

40

Multiplicity exercise

How many
- Employees can a Company have?
- Employers can a Person have?
- Owners can a BankAccount have?
- Operators can a BankAccount have?
- BankAccounts can a Person have?
- BankAccounts can a Person operate?



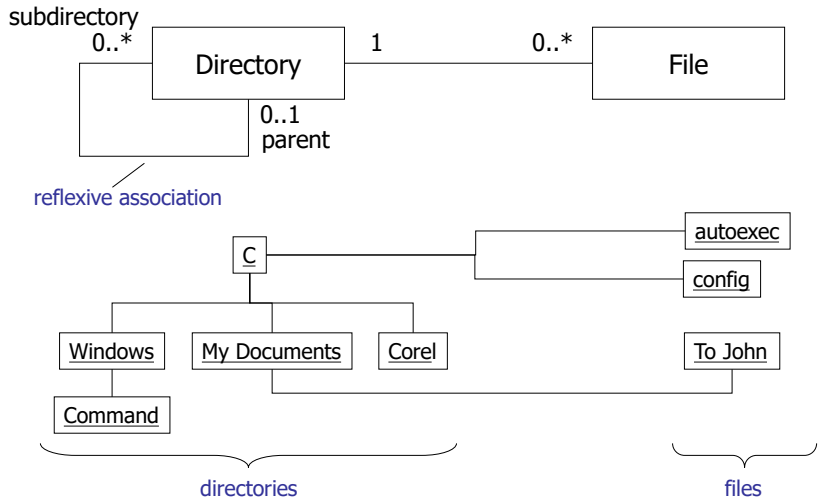
Exercise

Model a computer file system. Here are the minimal facts you need:
- The basic unit of storage is the file
- Files live in directories
- Directories can contain other directories

Use your own knowledge of a specific file system (e.g. Windows 95 or UNIX) to build a model

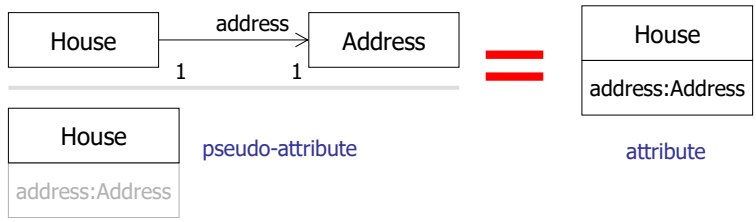
Hint: a class can have an association to itself!

Reflexive associations



43

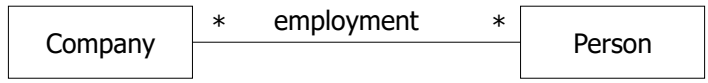
Associations and attributes



- If a navigable relationship has a role name, it is as though the source class has a pseudo-attribute whose attribute name is the role name and whose attribute type is the target class.
- Objects of the source class can refer to objects of the target class using this pseudo-attribute.
- Use associations when:
  - The target class is an important part of the model.
  - The target class is a class that you have designed yourself and which must be shown on the model.
- Use attributes when:
  - The target class is *not* an important part of the model e.g. a primitive type such as number, string etc.
  - The target class is just an implementation detail such as a bought-in component or a library component e.g. java.util.Vector (from the Java standard libraries).

44

Association classes

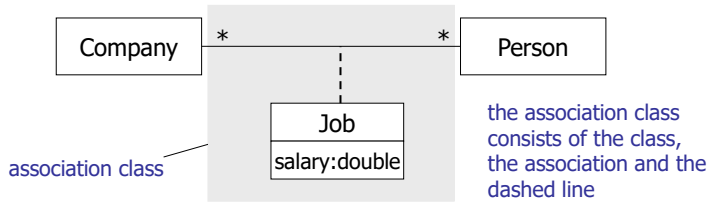


Each Person object can work for many Company objects.
Each Company object can employ many Person objects.
When a Person object is employed by a Company object, the Person has a salary.

But where do we record the Person's salary?

- Not on the Person class - there is a different salary for each employment
- Not on the Company class - different Person objects have different salaries
- The salary is a property of the employment relationship itself
  - every time a Person object is employed by a Company object, there is a salary

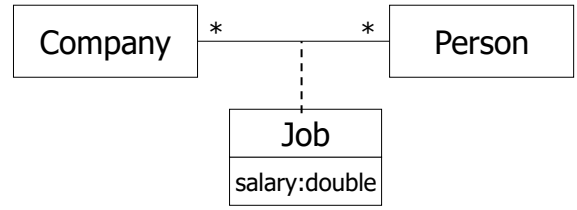
Association class syntax



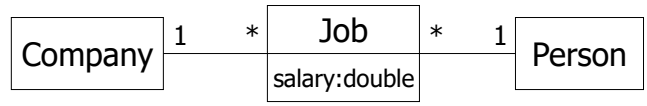
- We model the association itself as an association class. One instance of this class exists for ***each link*** between a Person object and a Company object
  - Instances of the association class are links that have attributes and operations
  - Can only use association classes when there is *one unique link* between two specific objects. This is because the identity of links is determined exclusively by the identities of the objects on the ends of the link
- We can place the salary and any other attributes or operations which are really features of the association into this class

Using association classes

If we use an association class, then a particular Person can have only *one* Job with a particular Company



If, however, a particular Person can have *multiple* jobs with the *same* Company, then we must use a *reified association*



Takeaway Messages (Part 3)

- UML Class diagrams include classes and relationships between classes
- The level of detail in modelling classes differs between analysis and design class diagrams:
  - Classes in ***analysis*** class diagrams represent abstractions from the ***problem domain only*** with attributes characterizing the real objects ignoring details important in implementation of classes with a programming languages (e.g., visibility, constructors, return types and parameters of operations)
  - Classes in ***design*** offer greater detail – capture the ***problem domain*** and ***the implementation domain***. (user interface, storage, communication, etc.).
    - Design classes must include details important for their implementation with an OO programming language.
- Relationships link two or more modelling elements
  - Relationships between objects are called ***links***
    - Objects can only exchange messages over links (between the communicating objects).
  - Relationships between classes are called ***associations***
    - Associations in analysis class diagrams have names, direction and multiplicities.

Further Reading (Part 3)

Jim Arlow's book "UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design", Edition 2: Chapter 9.

49

Part 4: Inheritance and polymorphism

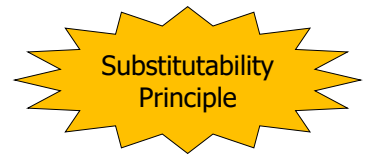
50

Generalisation

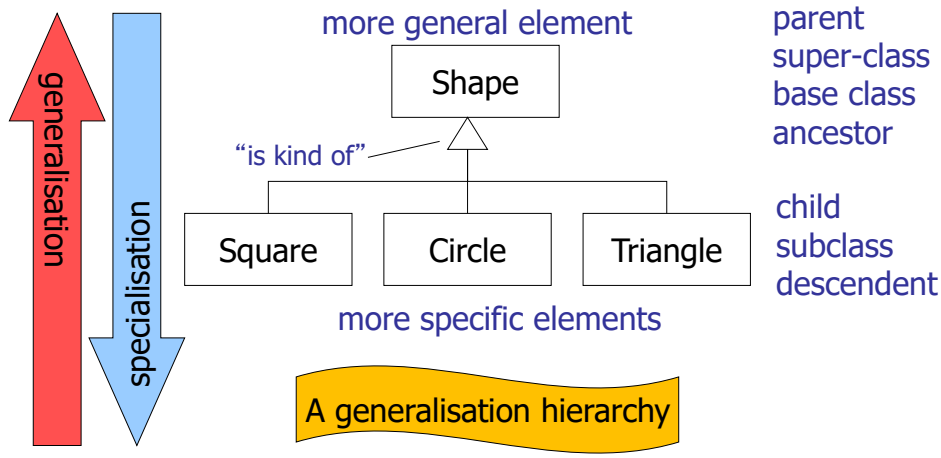
A relationship between a more general element and a more specific element

The more specific element is entirely consistent with the more general element but contains ***more*** information.

An instance of the more specific element may be used where an instance of the more general element is expected.



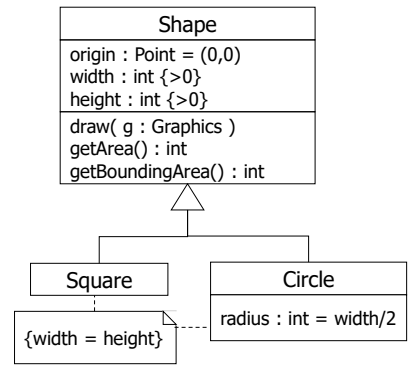
Example: class generalisation



Class inheritance

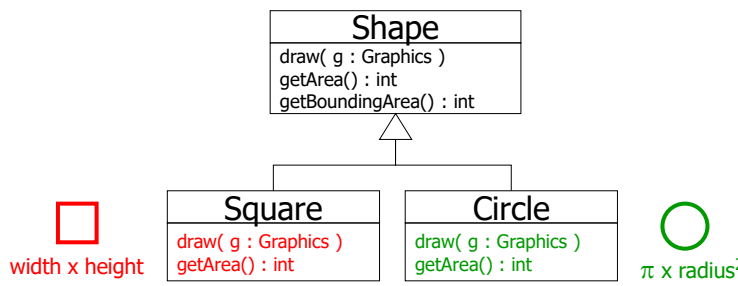
- Subclasses inherit *all* features of their super-classes:
  - attributes
  - operations
  - relationships
  - stereotypes, tags, constraints
- Subclasses can add new features
- Subclasses can ***override*** super-class operations
- We can use a subclass instance *anywhere* a super-class instance is expected.

Substitutability Principle



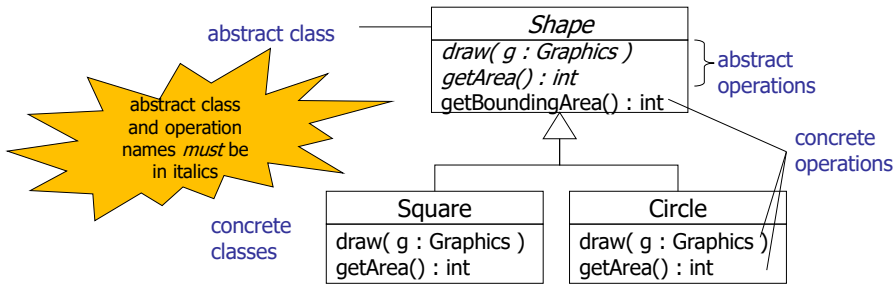
But what's wrong with these subclasses

Overriding



- Subclasses often need to *override* super-class behaviour
- To override a super-class operation, a subclass must provide an operation with the ***same signature:***
  - The operation signature is the operation name, return type and types of all the parameters
    - The names of the parameters don't count as part of the signature

Abstract operations & classes

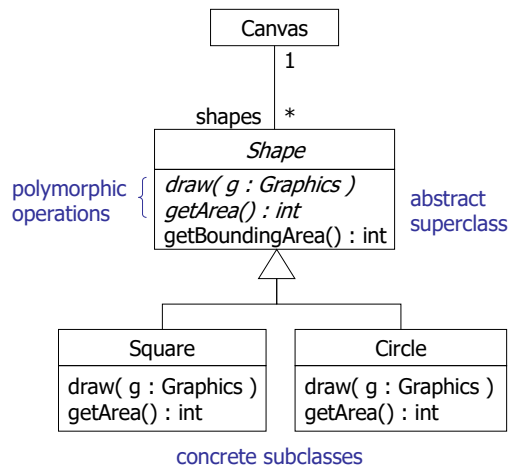


- We can't provide an implementation for *Shape :: draw(g : Graphics)* or for *Shape :: getArea() : int* because we don't know how to draw or calculate the area for a "shape"!
- Operations that lack an implementation are *abstract operations*
- A class with any abstract operations *can't* be instantiated and is, therefore, an *abstract class*

Polymorphism

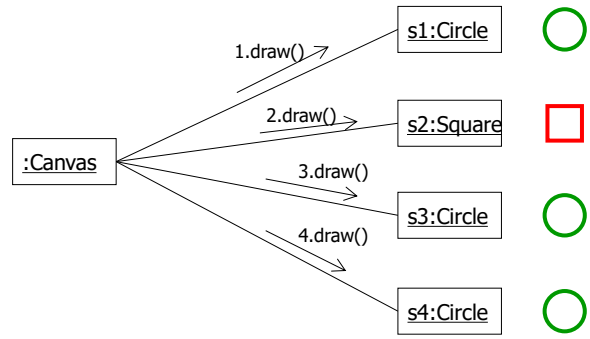
- Polymorphism = "many forms"
  - A polymorphic operation has many implementations
  - Square and Circle provide implementations for the polymorphic operations *Shape::draw()* and *Shape::getArea()*
- All concrete subclasses of Shape *must* provide concrete draw() and getArea() operations because they are abstract in the super-class
  - For draw() and getArea() we can treat all subclasses of Shape in a similar way - we have defined a contract for Shape subclasses

A Canvas object has a collection of *Shape* objects where each *Shape* may be a Square or a Circle

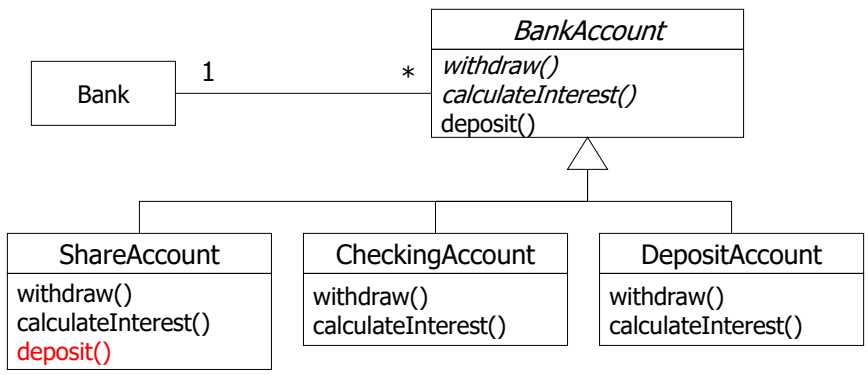


What happens?

- Each class of object has its own implementation of the draw() operation
- On receipt of the draw() message, each object invokes the draw() operation specified by its class
- We can say that each object "decides" how to interpret the draw() message based on its class



BankAccount example



- We have overridden the deposit() operation even though it is *not* abstract. This is perfectly legal, and quite common, although it is generally considered to be bad style and should be avoided if possible

Takeaway Messages (Part 4)

- Inheritance between classes is an essential OO concept
  - Sub-classes may have ***additional*** attributes and operations
  - Sub-classes can ***override*** operations
- Abstract classes contain at least one abstract operation, e.g., an operation which is not provided with an implementation code.
  - A class with an abstract operation is an abstract class and *cannot be instantiated*
- Polymorphism is achieved when multiple sub-classes implement the same operation of the superclass, but the implementations are different.

Further Reading (Part 4)

Jim Arlow's book "UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design", Edition 2: Chapter 10.

Part 5: Noun/Verb analysis in practice

61

Noun/Verb analysis: Worked example

We will look at the vision statement for an example case study, ***Town Map***

Note that we could also use other documents as input to this activity, and we would process them in exactly the same way:

- Use cases
- Project glossary
- Initial statement of the requirements (i.e. an Informal descriptions of the system)

62

Town map example

The system you are analysing is an electronic town map in a public car park that is designed to show the key points of the town.
The map has a light to identify each key location, such as the hospital and the library. Below the map is a panel containing buttons labelled with the location names. When a visitor pushes the button on the panel the light at the related location lights up for 10 seconds.
A visitor who wants a printed copy of the map can push the "print map" button, and a hard copy map is released through the map slot.
The system keeps a tally of the number of printed maps requested. If the machine runs out of paper, a sign saying "Sorry, no printed maps available" is lit up, until an operator comes and restocks with paper. If the paper gets low (less than about 100 sheets) an internal "paper low" light illuminates.
When an operator comes to maintain the TownMap they perform two tasks:
1) Restock with paper if necessary.
2) Press the internal "test lights" button to illuminate all the lights for 1 minute. If any of the lights have burned out, the operator replaces them

Highlight nouns & verbs

The system you are analysing is an electronic town map in a public car park that is designed to show the key points of the town.
The map has a light to identify each key location, such as the hospital and the library. Below the map is a panel containing buttons labelled with the location names. When a visitor pushes the button on the panel the light at the related location lights up for 10 seconds.
A visitor who wants a printed copy of the map can push the "print map" button, and a hard copy map is released through the map slot.
The system keeps a tally of the number of printed maps requested. If the machine runs out of paper, a sign saying "Sorry, no printed maps available" is lit up, until an operator comes and restocks with paper. If the paper gets low (less than about 100 sheets) an internal "paper low" light illuminates.
When an operator comes to maintain the town map they perform two tasks:
1) Restock with paper if necessary.
2) Press the internal "test lights" button to illuminate all the lights for 1 minute. If any of the lights have burned out, the operator replaces them.

Noun list

system	panel
town map	light
public car park	location
map	map slot
light	sign saying "Sorry, no printed maps available"
key location	operator
hospital	paper
library	internal "paper low" light
map	operator
panel	tally
buttons labelled with the location	town map
names	lights
visitor	operator
button	

Notice that there will usually be:

- Duplicates
- Synonyms – different words have the same meaning e.g. "map" and "town map"
- Homonyms – the same word has different meanings e.g. "button" – there are two different types of button

Noun phrase *analysis* (an illustration)

System – (Class? Attribute? Neither?) – Neither, ignore

town map - (Class? Attribute? Neither?) – Neither, ignore (but see below map)

public car park - (Class? Attribute? Neither?) – Neither, ignore

(town) map - (Class? Attribute? Neither?) – Class TownMap

Location button - (Class? Attribute? Neither?) – Class LocationButton

key location - (Class? Attribute? Neither?) – Attribute of LocationButton

...

Refined noun list

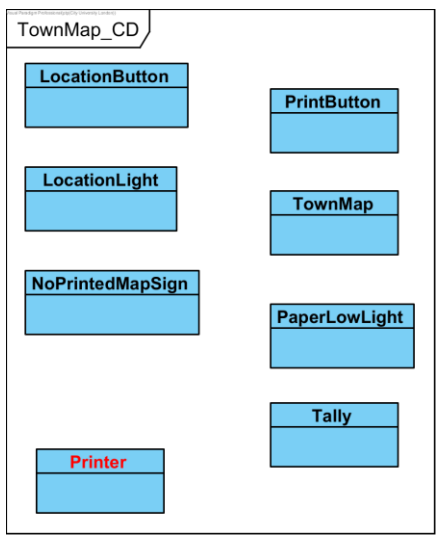
hospital
library
location
location button
location light
map slot
operator (actor?)
paper
paper low light
print map button
public car park
sign saying "Sorry, no printed maps available"
system
town map
tally
visitor (actor?)

Nouns are ordered alphabetically.

We have:

- Removed duplicates
- Resolved synonyms and homonyms
- Identified nouns that don't seem to be *part of the system (in red)*
- Identified possible ***actors***

Candidate classes



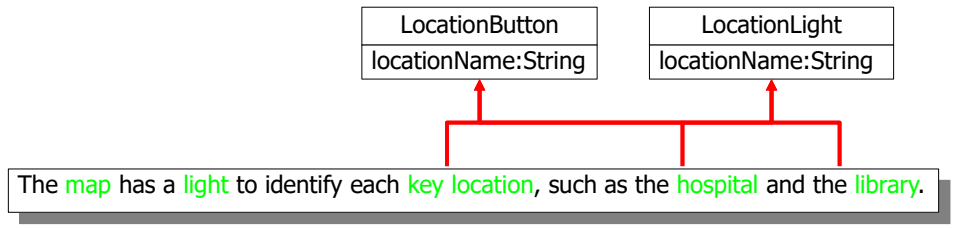
We have analysed the list of nouns to come up with some candidate classes

- Candidate classes are classes that represent possible analysis classes

We have applied naming standards (UpperCamelCase) to give the classes good names

Some classes are still missing – e.g. what prints the maps?

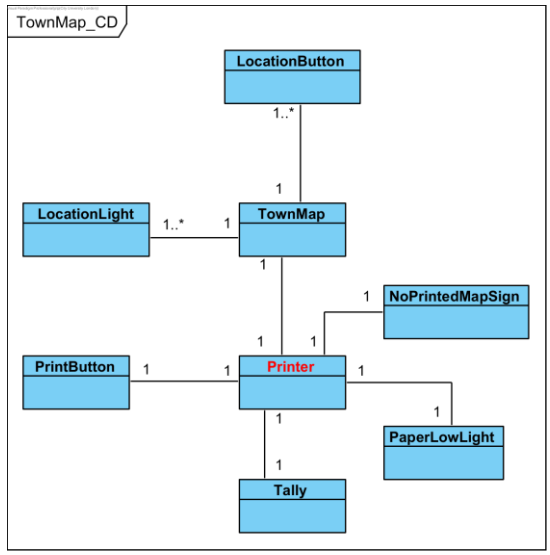
Find attributes



Nouns also indicate attributes of candidate classes

- Look for nouns that might indicate *parts of something*
- Look for nouns that refer to simple things such as *names, numbers and dates* – such things are usually attributes (of classes)

Organizing the candidate classes



The next step is to create a class diagram showing the candidate classes and how they ***might*** be related

Consolidate Verb list

show the key points of the town
location lights up for 10 seconds
push the "print map" button
a hard copy map is released through the map slot
system keeps a tally of the number of printed maps requested
the machine runs out of paper
sign saying "Sorry, no printed maps available" is lit up
until an operator comes and restocks with paper
paper gets low (less than about 100 sheets)
internal "paper low" light illuminates
Restock with paper if necessary
Press the internal "test lights" button to illuminate all the lights for 1 minute
lights have burned out
operator replaces them

We have to convert these verb phrases into operations of the candidate classes or use to name the associations.

Apply Verb analysis

show the key points of the town – (Operation? Association? Neither?) –

Operation of TownMap: turnOn()
turnOff()

location lights up for 10 seconds – (Operation? Association? Neither?)

Operation of LocationLight: turnOn()
turnOff()

push the "print map" button – (Operation? Association? Neither?)

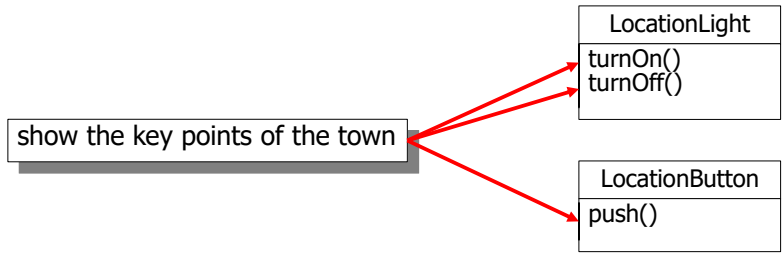
Operation of LocationButton: push()

….

Restock with paper if necessary - (Operation? Association? Neither?)

Neither, ignore

Finding operations

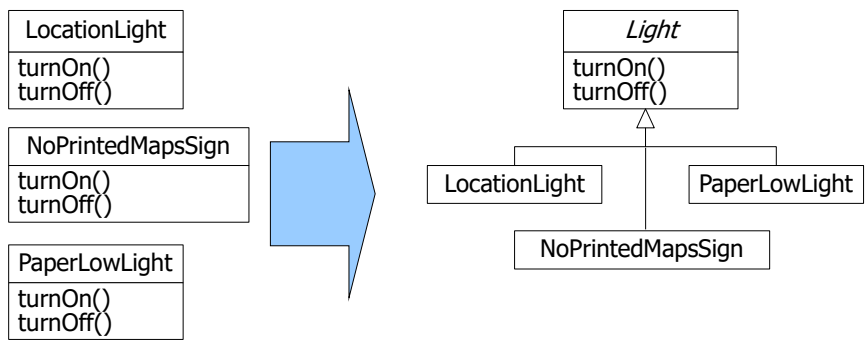


You must resolve each verb phrase into one or more operations on your candidate classes

These operations must be distributed amongst the candidate classes in a way that ***makes sense*** from a business perspective

You may have to introduce ***new candidate classes*** to do this!

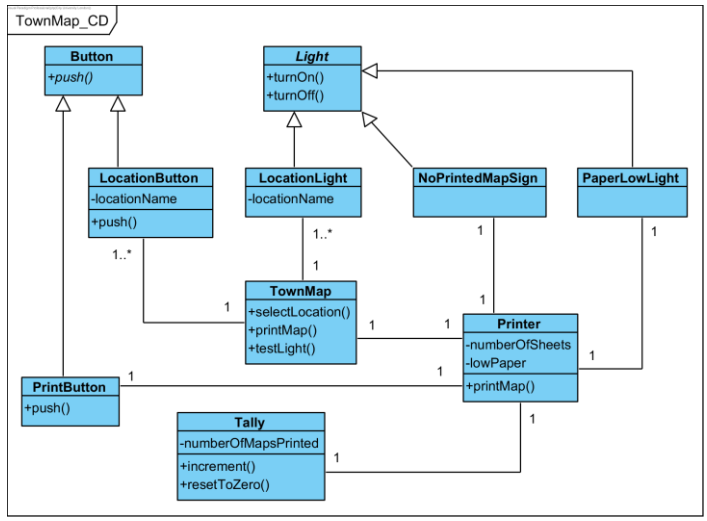
Look for possible generalizations



Classes that have attributes and operations in common *might* be part of a generalization hierarchy

- Only use generalization in analysis if it improves the business semantics of the model – this can be subjective!

Putting it all together



The result of this process is what we call a "first cut" analysis class diagram

- It's "first cut" because there is still more analysis work to be done!
- Diagram can be *refined* applying other methods of analysis.
- Add *association names, roles, direction*.

75

Summary (1)

We have seen a specific technique to move from textual descriptions (e.g., use case specifications and requirements) to analysis classes:

- Noun verb analysis. This is a systematic (almost mechanistic) process of analysing the text.
  - A noun can be:
    - A class from the problem domain
    - An attribute of a class
    - Neither, hence should be ignored and not included in the class diagram. *Actors* are often ignored (as they are outside the system) unless the system stored details about them (username/password/role)
  - A verb can be:
    - An operation of a class
    - An association between two classes
    - Neither (e.g. a generic verb) and should be ignored.
- Noun/verb analysis does not guarantee that all important classes will be captured, but is a good starting point for analysis.

We have looked at relationships between classes/objects:

- Associations
- Associations classes
- Inheritance and the related concept of polymorphism

76

Summary (2)

- Structural models represent the objects that are used and created by a business system
- Object oriented approach
  - Describing a system as made up of objects
  - The structure of software systems is ***specified*** by class diagrams (classes and their relationships)
- *Objects*: encapsulation of data and operations
- *Classes*: types of objects, an objects is an instance of a class
  - Defined through their attributes, operations
- We have seen a specific technique to move from descriptions (use cases and requirements) to analysis classes:
  - Noun verb analysis

- Readings for this lecture and the next:
  book chapter 6 – 10
- Next Lecture: Advanced Class Diagrams