EBU5304 – Software Engineering

Analysis and Design

- Analysis
  - Purpose of Analysis
  - Stereotypes of classes
  - Class relationships
- Design
  - Purpose of Design
  - Design principles
  - Design quality
  - Class design

Analysis

What is analysis?

- "A method of studying the nature of something or of determining its essential features and their relations".
- "A method of exhibiting complex concepts or propositions as compounds or functions of more basic ones".
- "The evaluation of an activity to identify its desired objectives and determine procedures for efficiently attaining them".

http://www.dictionary.com

What makes something essential?

We can't take the time to go through <u>all</u> of the possibilities.

No, but we can identify the <u>essential</u> features and relations.

This is an Intentionally BLANK slide. We will discuss it in class and you can write down notes.

1 _____

2 _____

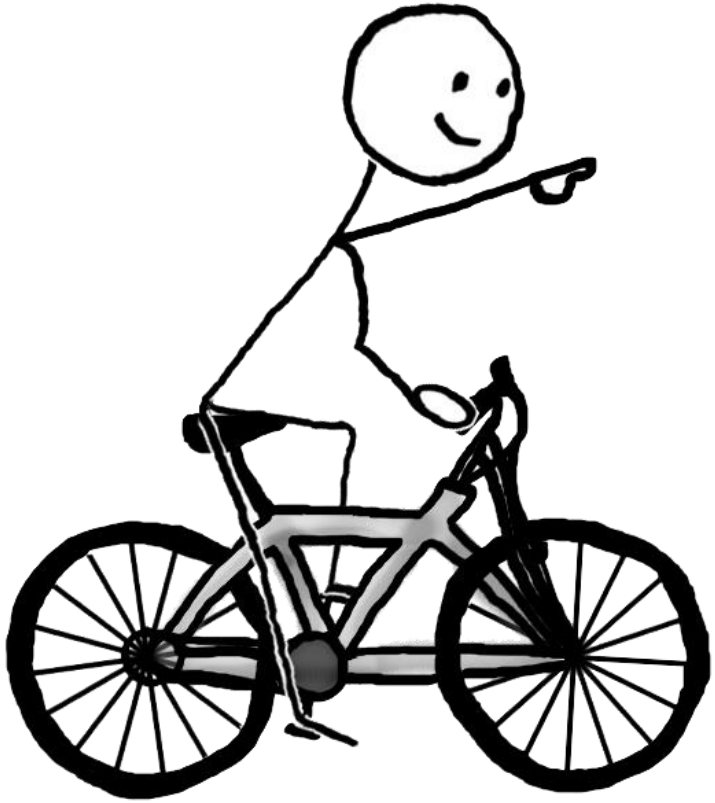
3 _____

4 _____

Why didn't the customer tell you?

Possible reasons

- They may assume you know about a feature because it's obvious to them.
- They may not think about some of the special conditions where the feature would be needed.
- They may not know it's necessary!



Getting to the essentials

Gather requirements → Analyse in real world context → Develop the architecture

- Textual analysis
  - Nouns in requirements and documents.
- Entities and concepts
  - From the application domain.
- Experience
  - Previous systems.

Why "Analyse"

- Focus shifts to developer and system internals.
  - Refining requirements.
- Aim: precise understanding of requirements.
  - Process of structuring requirements:
    - Understand
    - Change
    - Reuse
    - Maintain

What concerns Analysis

- To be used mainly by developers
  - Using the “language” of the developer.
- Provide internal view of the system.
- Conceptual modelling
  - Structured by stereotypical classes and packages.
- Function realisation
  - Outlines how to realise the functionality within the system.

Conceptual modelling

- A conceptual model aims to identify the individual concepts (classes) which exist within a problem domain.
- It should show: (Object Oriented Analysis)
  - Concepts (fundamental classes)
  - Attributes of concepts
  - Operations of concepts (leave details to the design stage)
  - Associations between concepts
- Conceptual models are described using Class diagrams.

Analysis Class (1/2)

- Objects are entities that model some concrete or conceptual entity inside the system.
  - A class is an abstraction of an object.
  - Every object belongs to a class, and the class of an object determines its interface (outside world view of the object).
  - The process of creating a new object belonging to a particular class is called instantiating or creating an instance of the class.

Analysis Class (2/2)

- Analysis classes are conceptual:
  - High level behaviour
  - High level attributes
  - High level relationships and special requirements
- Analysis classes always fit in one of 3 basic stereotypes
  - Entity classes
  - Boundary classes
  - Control classes

Entity classes

- Entity classes
  - Used to model information that is long-lived and persistent
    - Logical data structure
  - Information that the system is dependent on.

Boundary classes

- Boundary classes
  - Used to model the interaction.
  - Often involve receiving (presenting) information and requests from (and to) users and external systems.
  - Normally represent abstractions of windows, forms, communication interfaces, printer interfaces, sensors, terminals, etc.
  - The analysis boundary classes do not describe how the interaction is physically realised, only what is achieved by the interaction.

Control classes

- Control classes
  - Used to encapsulate control and coordination of the main actions and control flows.
  - Represent coordination, sequencing, transactions and control of objects.

Attributes

- Attributes are descriptions of a particular data item maintained by each instance of a class.
- Every attribute has a name, a type, and if required a default initial value.
  - During analysis, the attribute name and type can be abstract, for example: *account name*, *string*.
  - During later design, they should have the syntax of the target language, for example: *accountName*, *String*.
- Attributes should be documented with clear, concise definitions.

Operations

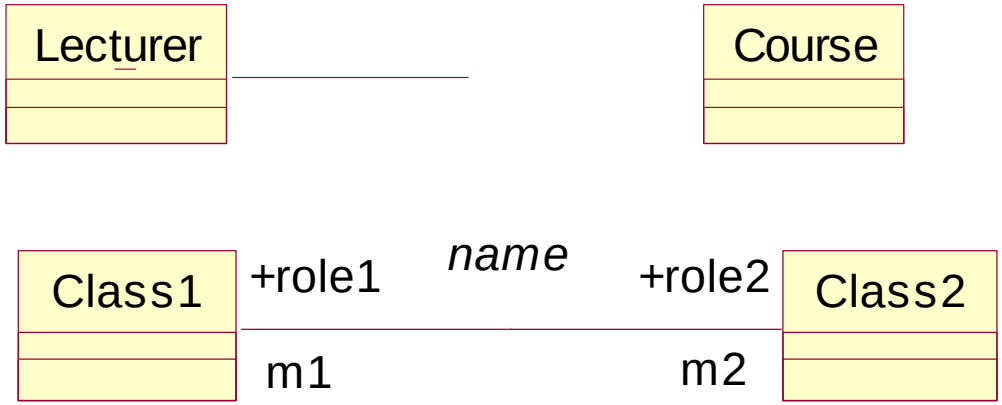
- Operations are abstract specifications of a class's behaviour.
- They have a name, a set of input parameters, and a return type.
  - Details of the functionality of an operation are specified textually.
- An operation should only do one thing:
  - Methods implement operations.
  - Operations should be documented to state the functionality performed by the operation.

Class Relationships

- A system is made up of many classes and objects. Relationships provide the pathway for communication.
- Relationships
  - Association
  - Inheritance
    - Generalisation
    - Specialisation

Association

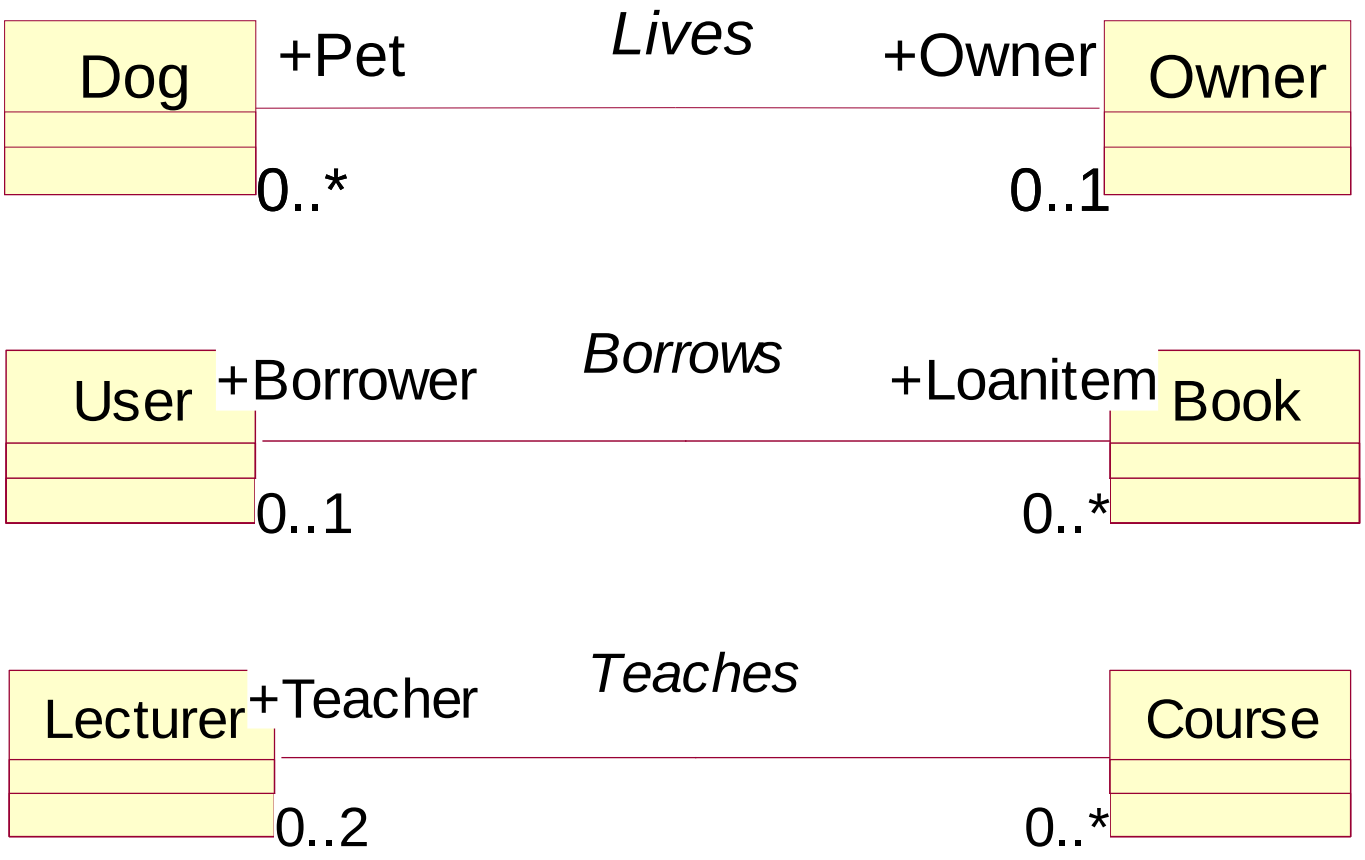
- An association is a bidirectional semantic connection between classes:
  - Data may flow in either direction.
- An association means there is a link between objects.



Association

- Associations have:
  - a name, the meaning;
  - a role, which describes the role the instances of the associated class play in the relationship;
  - a multiplicity which states how many instances of a class at one role end can be associated with an instance of another class at the other role end.

Association Examples



Association Multiplicity

- Multiplicity indicators
  - 1 Exactly one
  - 0..* Zero or more
  - 1..* One or more
  - 0..1 Zero or one
  - 5..8 Specific range (5, 6, 7, 8)
  - 4..7, 9 Combination range (4,5,6,7 or 9)

Association Multiplicity

- Examples:

An A is associated with exactly one B.

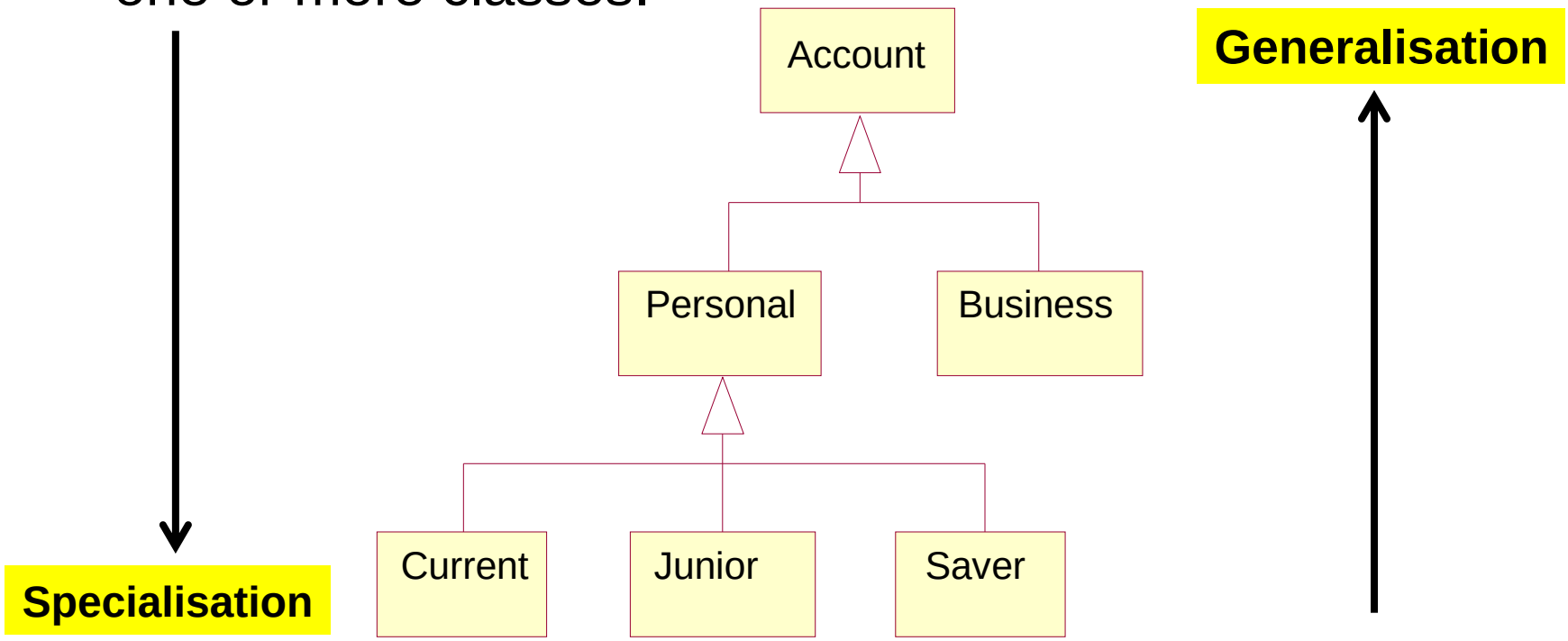
An A is associated with one or more B.

An A is associated with zero or one B.

An A is associated with zero or many B.

Inheritance (1/2)

- Inheritance defines a relationship among classes where one class shares the attribute(s) and/or operation(s) of one or more classes.



Inheritance (2/2)

- "is-a", "kind-of" hierarchy
- A subclass will inherit all attributes, operations, relationships defined in any of its superclasses.
- Subclass may be augmented with additional attributes and operations.
- Subclass can override attributes and operations.
- The key to reuse.

Analysis steps

Activities:

1. Identify Entity, Boundary and Control classes
2. Identify class relationships
3. A conceptual class diagram
4. Identify attributes for each entity class
5. Add constraints

Please go through the activities in the case study.

Design

What is design?

I thought we were doing design already? Is there really a big difference between analysis and design? When can we get to the coding?

Blueprint

Plan

Structure

The purpose for something

Style

I looked up “design” on the Web and found a lot of different definitions. Which is right?

Common design characteristics

- Designs have a purpose
  - They describe HOW something will work.
- Designs have enough information so that someone can implement them.
- There are different styles of design
  - Like different types of house architectures.
- Designs can be expressed at different levels of detail
  - A dog house needs less detail than a skyscraper.

Our definition of "design"

- Software design is the process of planning how to solve a problem through software.
- A software design contains enough information for a development team to implement the solution. It is the embodiment of the plan
(i.e. the *blueprint for the software solution*).

Role of Design

- Design transforms the analysis model into a design model that serves as a blueprint for software construction.
- At this point, consideration needs to be taken for the non-functional requirements e.g.
  - The programming language chosen
  - Operating systems
  - Databases
  - User-interfaces
- During the design phase: break down the overall task.
- Create a ‘skeleton’ of the system that the implementation can easily fit into.

Design Quality Guidelines

- A good software design should:
  - Meet the requirements
  - Be well structured: exhibit an architecture
  - Be modular
  - Contain distinct representations of data, architecture, interfaces, and components
  - Be maintainable
  - Be traceable
  - Be well documented: represented using a notation that effectively communicates its meaning
  - Be efficient (when implemented)
  - Be error free

Fundamental Concepts

- Abstraction: data, procedure, control
- Architecture: overall structure of the software
- Patterns: a proven design solution
- Modularity: compartmentalization
- Information hiding: encapsulation
- Functional independence: coupling and cohesion
- Refinement: elaboration of detail for all abstractions
- Refactoring: a reorganization technique that simplifies the design

Coupling

- The number of dependencies between subsystems.
- Indicates strengths of interconnections
  - Tight: relatively dependent. Modifications to one is likely to have impact on others.
  - Loose: relatively independent. Modifications to one will have little impact on others.
- Ideally, subsystems are as loosely coupled as reasonable …
  - to minimise the impact on errors or future change.

Cohesion

- The number of dependencies within a subsystem.
- A measure of the level of functional integration within a module.
  - High: objects are related to each other and perform similar tasks.
  - Low: unrelated objects.
- Ideally, a subsystem should have high cohesion.
  - All parts of the component should contribute to its logical function.
  - If it is necessary to change the system, then everything to do with the component is encapsulated in one place.

Advantages of Object Oriented Design

- Easier maintenance:
  - Objects are independent.
  - Objects may be understood as stand-alone entities.
- Objects are potentially reusable components:
  - Reuse previous developed objects
  - Standard object
  - Inheritance
- For some systems, there may be an obvious mapping from real world entities to system objects.

Object-oriented Architecture

- OO principles:
  - Abstraction
  - Encapsulation
  - Inheritance
  - Polymorphism
- OO patterns
  - Will be introduced in later lectures in week 3 and 4

Design a class (1/4)

- Classes defined with:
  - Outlining the Design class
  - Identifying Operations and Attributes
  - Identifying Relationships: Associations / Generalisations
  - Describing Methods
- Classes:
  - Boundary classes: dependent on the *specific interface technologies* in use.
  - Entity classes: represent *persistent information*; usually created using databases.
    - Example: Creating design classes that map to tables in a relational data model.
  - Control classes: *distribution issues*, *performance issues*, *transaction issues*.

Design a class (2/4)

- Identifying Operations
  - Identify the operations that need to be provided by the design class and describe those operations using the syntax of the chosen programming language.
  - Inputs
    - Responsibilities of associated analysis classes
      - If inputs and outputs are described for the responsibilities, they can be used as a first outline of formal parameters and result values of operations.
    - Special requirements
      - For example, incorporating some generic design mechanisms or technology such as database technology.
    - Interfaces that class needs to provide.

Design a class (3/4)

- Identifying Attributes
  - Identify the attributes required by the design class and describe them using the syntax of the chosen programming language.
  - Guidelines
    - Consider the attributes on any related analysis class.
    - The available attribute types are restricted by the programming language.
    - Try to use existing attribute types.
    - A single attribute cannot be shared by several design objects; *if this is required*, the attribute needs to be defined as a separate class.
    - *If the class becomes too complicated because of its attributes*, some attributes may be separated into classes of their own.

Design a class (4/4)

- Identifying Associations
  - Guidelines
    - Consider corresponding analysis classes.
    - Refine association multiplicities, role names, association classes etc. in line with programming language rules.
    - Refine the navigability of associations; the direction of message transmissions between design objects implies corresponding navigabilities of associations between their classes.
  - Describing methods
    - Can be used during design to specify how operations are realised.
    - Can be specified using natural language or pseudo-code.
    - Usually created during implementation rather than design.

Design steps

Activities:

1. Based on the conceptual class diagram produced from the Analysis stage.
2. Identifying Class Relationships: Associations / Generalisations
3. Identify operations
4. Describing methods
5. Captures implementation requirements.
6. Produce detailed design class diagram.

Please go through the activities in the case study.

Summary

- Analysis
  - Purpose of Analysis
  - Stereotypes of classes
  - Class relationships

- Design
  - Purpose of Design
  - Design principles
  - Design quality
  - Class design

References

- Chapter 4, 5 – “Head First Object Oriented Analysis & Design” textbook by Brett McLaughlin *et al*
- Chapters 6, 7 – “Software Engineering” textbook by Ian Sommerville