SPE 2017 – Lecture 13

Object Oriented Analysis and Design

Part A: Domain Modelling

Dr. Daniel Schien

Daniel.schien@bristol.ac.uk

Recap

- Week 1 Introduction & The Open Project
- Week 2 Introduction to Agile & Agile Practices
- Week 3 CI & Validation and Verification
- Week 4 Requirements I & Requirements II
- Week 5 OOAD I & II
- Week 6 Spring & CD

Why Object Oriented Programming?

- Abstraction
- Object vs Functions
- Reuse
- Extensibility

Object Oriented Analysis and Design

- Analysis
  - Domain Modelling
- Design
  - Static properties
  - Dynamic properties
  - Architecture

Domain Modelling

Modelling in the SELC

- Distinguish domain and implementation/design models
- Domain models = conceptual model
- We ignore any constraints from the software implementation.
- We want to just be influenced by what's in the domain
- Understand, communicate, document

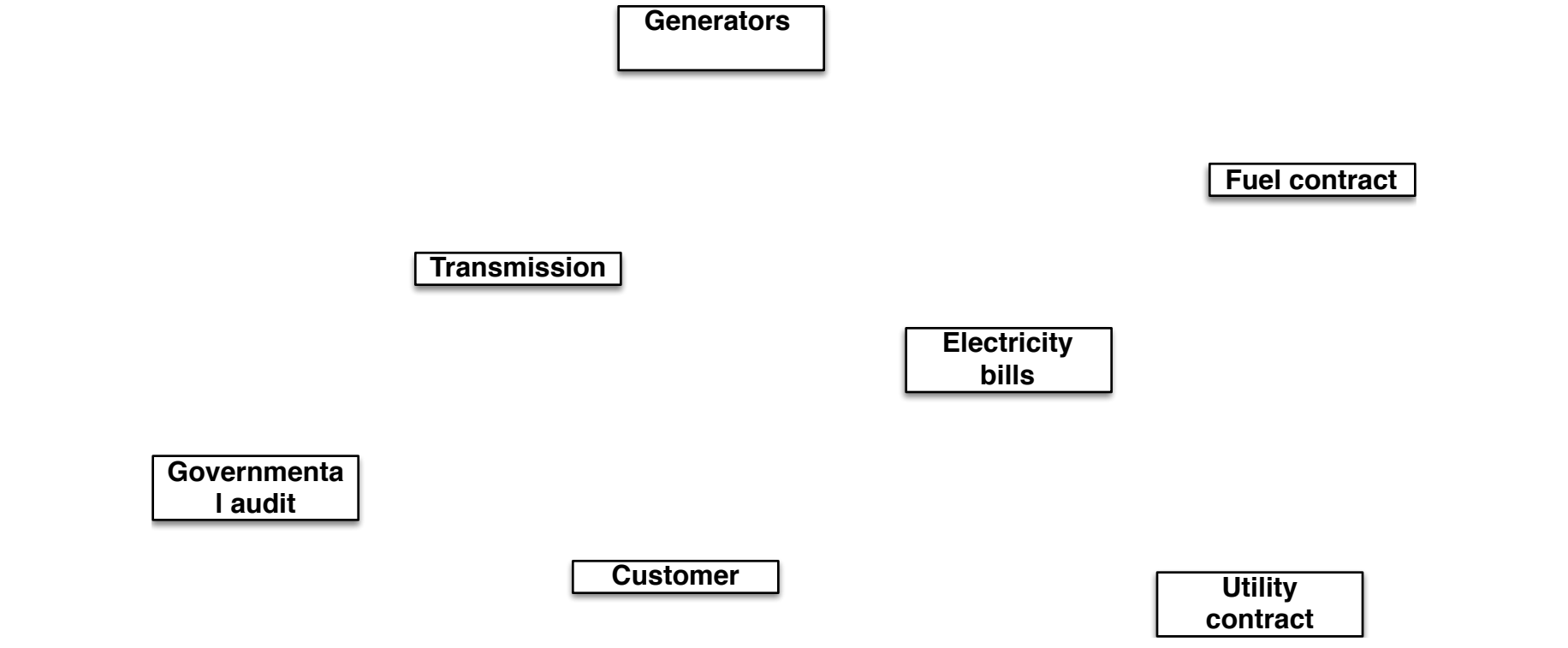
Identifying Domain Entities

- Not mechanical
- Discussion Stakeholders
- Things to look out for
  - Objects
  - Business process
  - Job titles /roles
  - Sets, grouping, containers, elements
  - Events
  - Record of work

Exercise – Tasked to Build a Management System for a DNO

Brainstorm entities in a model of the electrical grid

- Objects
- Business process
- Job titles /roles
- Sets, grouping, containers, elements
- Events
- Record of work



Grammatical Parse

- Identify nouns from existing text
- Narrow down to remove
  - Duplicated variations
  - Irrelevant
  - Out of scope

Example

"Cloud computing is an information technology (IT) paradigm, a model for enabling ubiquitous access to shared pools of configurable resources (such as computer networks, servers, storage, applications and services),[1][2] which can be rapidly provisioned with minimal management effort, often over the Internet." https://en.wikipedia.org/wiki/Cloud_computing

Example

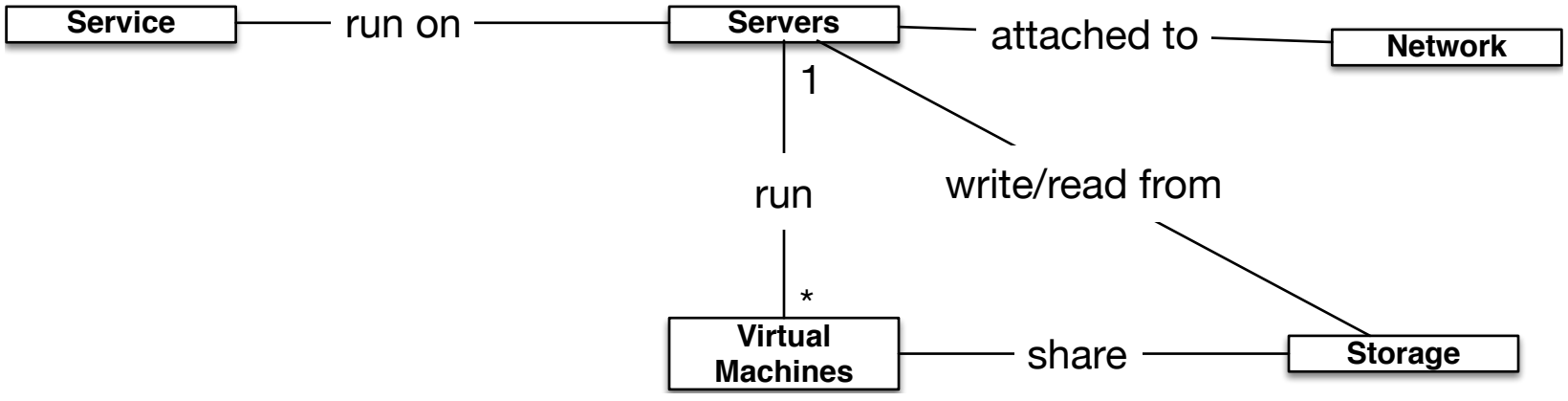
"Cloud computing is an information technology (IT) paradigm, a model for enabling ubiquitous access to shared pools of configurable resources (such as computer networks, servers, storage, applications and services),[1][2] which can be rapidly provisioned with minimal management effort, often over the Internet." https://en.wikipedia.org/wiki/Cloud_computing

Entity associations

- Describe relationships
- Look out for
  - Part of
  - Description for
  - Set member
  - Uses/manages
  - Communicates with
  - Owned by
  - Event

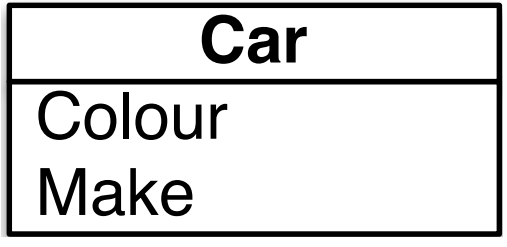
Associations

- Can be named
  - Association name
  - Role names
- Can include multiplicity
- There can be more then one between two entities
- They can be recursive



Attributes

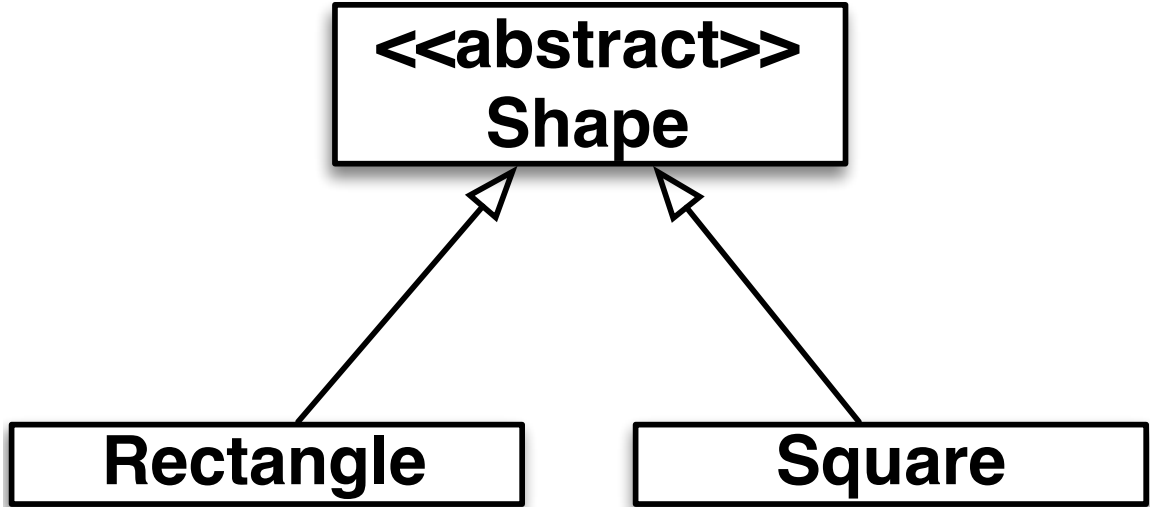
- Entity properties
- Distinguish from entities
  - Not sufficiently important to be modeled independently
  - Primitives (numbers, strings)
  - Immutable
- Identify - look for possession
  - e.g. “A car’s color”, “her name”



Generalisation

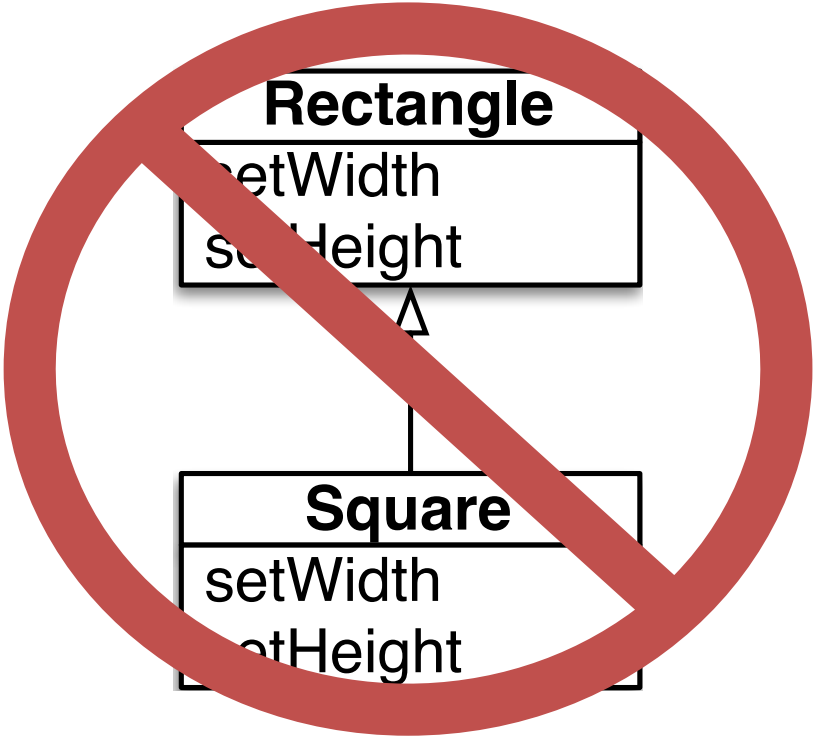
- Hierarchical relationships between entities
- *Subclasses* share common properties of *Superclass*
- Superclasses might only be abstractions from entities (e.g mamal) – stereotype <<abstract>>

Example



Liskov Substitution Principle

- If B is a subtype of A, then B should not alter any relevant properties of A.
- In the square the values of adjacent sides are not independent.



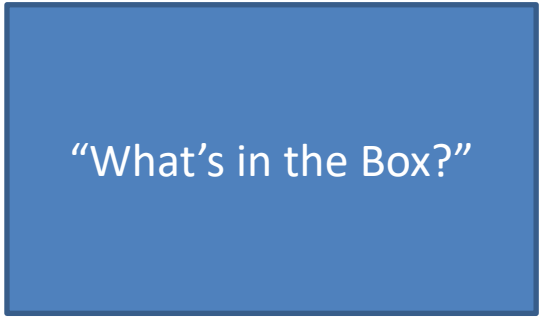
Notation

Just enough UML

- Modelling language
- Visualise, construct and document
- Standardised syntax
- Latest version 2.0

Notation - Domain Model

- The terminated bi-partite connector
  - Aka “The line”
- “The box”



Design

Structural Properties with Class Diagrams

Software Design and Implementation

- The point at which an executable software system is developed
- Creative activity
- Implicit or explicit
- OO design is expensive
- Agile approaches apply judgement
- OO designs are communication device

From Problem Space to Solution Space

- Problem Space = Domain Model
- Solution Space = Design Model

Domain Model

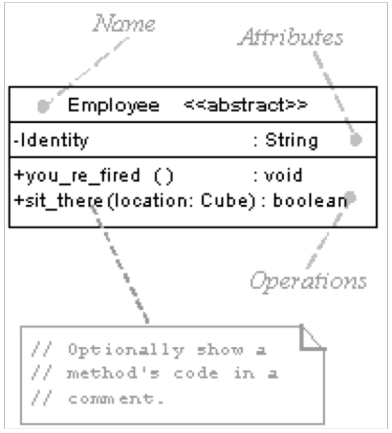
- Help understanding the domain
- Entities and relationships in the domain
- Contains out-of-scope elements

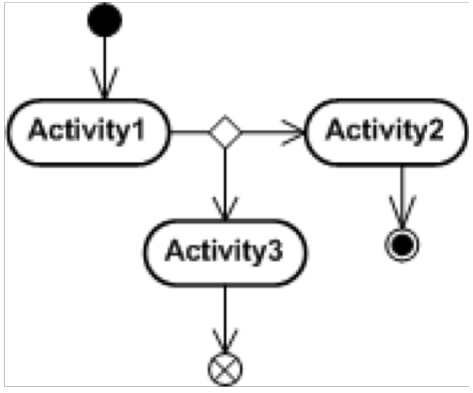
Design Model

- Structure the software system
- Software Classes with associations
- No out-of-scope elements
- Includes non-domain concepts

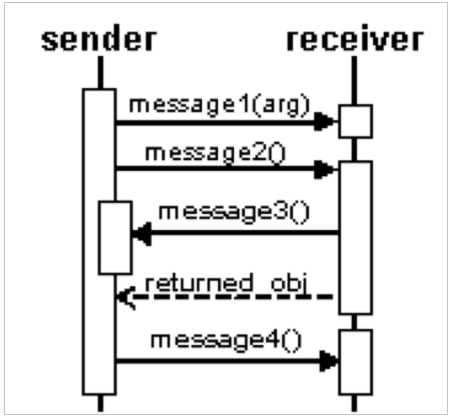
What are Design Models - exactly?

- Whatever works for you
- UML diagram types are useful





https://holub.com/uml/



Class Diagrams

- Visualises structure of a system
- Contains
  - system classes (entities)
  - their attributes
  - ~~operations (or methods)~~ (in design diagrams)
  - the relationships among objects

How do you construct Design Models?

- Whatever works for you
- Everybody/ team/ company/ author has a different process
- Start with the domain model
- Identify classes and associations
- Add behaviour (responsibility)
- Apply separation of concern
- Repeat

Let's repeat that

- Import your domain model entities as candidate classes
- Add any necessary associations and attributes
- Introduce generalizations
- Assign responsibilities
- Iterate

University of BRISTOL

CRC Cards

Class Responsibility Collaborator

A Laboratory For Teaching Object-Oriented Thir
Kent Beck, Apple Computer, Inc. Ward Cunningl
Wyatt Software Services, Inc.

A Laboratory For Teaching Object-Oriented Thinking

Kent Beck, Apple Computer, Inc.
Ward Cunningham, Wyatt Software Services, Inc.

It is difficult to introduce both novice and experienced procedural programmers to the anthropomorphic perspective necessary for object-oriented design. We introduce CRC cards, which characterize objects by class name, responsibilities, and collaborators, as a way of giving learners a direct experience of objects. We have found this approach successful in teaching novice programmers the concepts of objects, and in introducing experienced programmers to complicated existing designs.

1. Problem

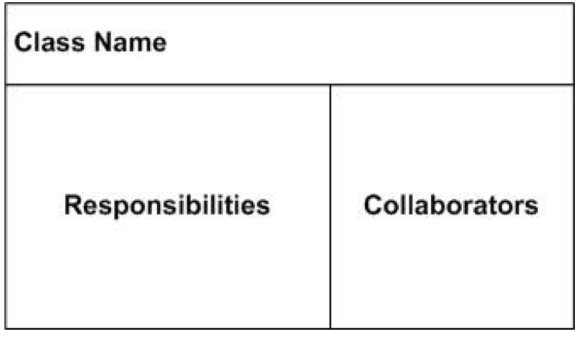
The most difficult problem in teaching object-oriented programming is getting the learner to give up the global knowledge of control that is possible with procedural programs, and rely on the local knowledge of objects to accomplish their tasks. Novice designs are littered with regressions to global thinking: gratuitous global variables, unnecessary pointers, and reduces to teaching the design of objects. We focus on design whether we are teaching basic concepts to novices or the subtleties of a complicated design to experienced object programmers.

Rather than try to make object design as much like procedural design as possible, we have found that the most effective way of teaching the idiomatic way of thinking with objects is to immerse the learner in the "object-ness" of the material. To do this we must remove as much familiar material as possible, expecting that details such as syntax and programming environment operation will be picked up quickly enough once the fundamentals have been thoroughly understood.

It is in this context that we will describe our perspective on object design, its concrete manifestation, CRC (for Class, Responsibility, and Collaboration) cards, and our experience using these cards to teach both the fundamentals

CRC Cards

- Class Responsibility Collaborator
- Team activity
- One CRC card for each relevant class
- Index cards
- Restricted Space
- Boundary object
- Analyse Use Cases or Validate Designs



a high level description of the purpose of the class

Other classes needed for responsibilities

Thursday:
Dynamic Properties with Activity Diagrams
Architectures

Thank you for your attention