Software Engineering
The University of Auckland

Quality Assurance
Data Modeling

Part II - Lecture 5

Today's Outline

- More about Modeling
- Data Modeling
- Creating Data Models with UMLet

Software Engineering
The University of Auckland

More about Modeling

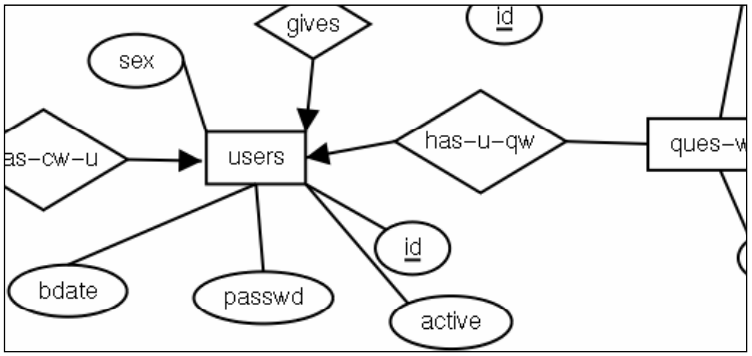
All models are wrong;
some models are useful
(George Box)

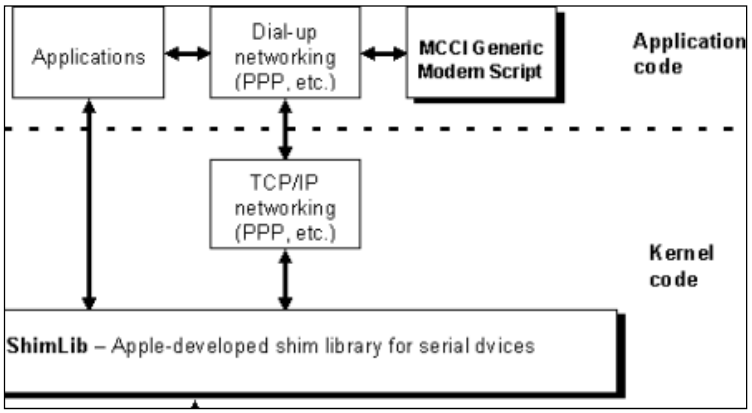
Models in SE

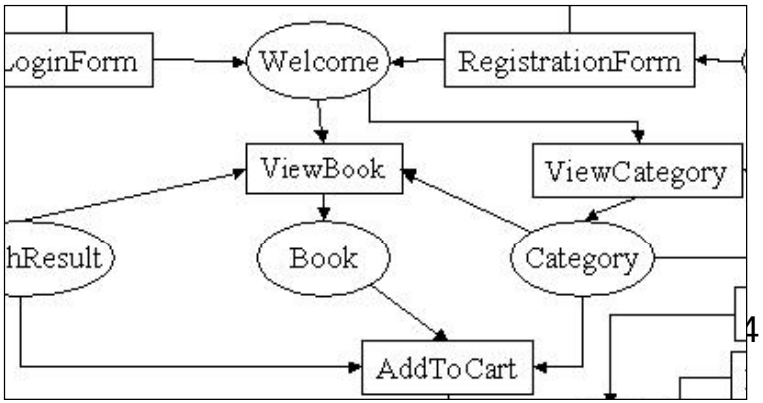
- **Data models**
  Data types and their relations
  E.g. ER-diagrams, class diagrams

- **Architecture models**
  Components of a system and their relations

- **User interface models**
  Structure of the UI (navigation, interaction, …)
  E.g. formcharts, screen diagrams

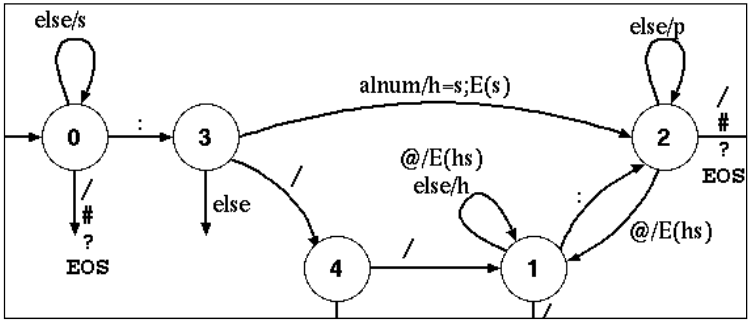




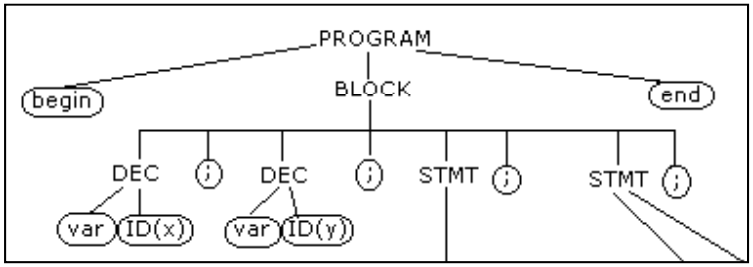


More Models in SE

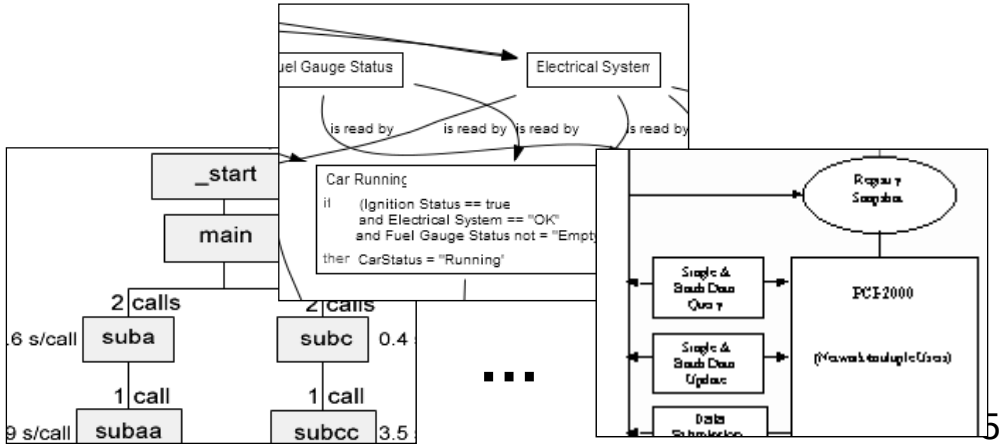
- **State transition models**
  System states and transitions
  E.g. lexical scanners, game state machines



- **Source code models**
  Structure of program code
  E.g. abstract syntax trees (AST)



- **Call graphs, dependency graphs, data flow diagrams & many more…**



Metamodels, Models and Model Instances

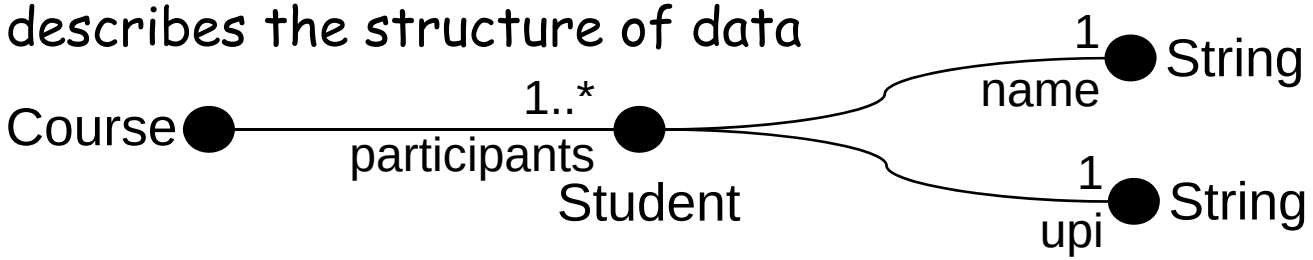
Software Engineering
The University of Auckland

Distinguish between model and model instance (data)

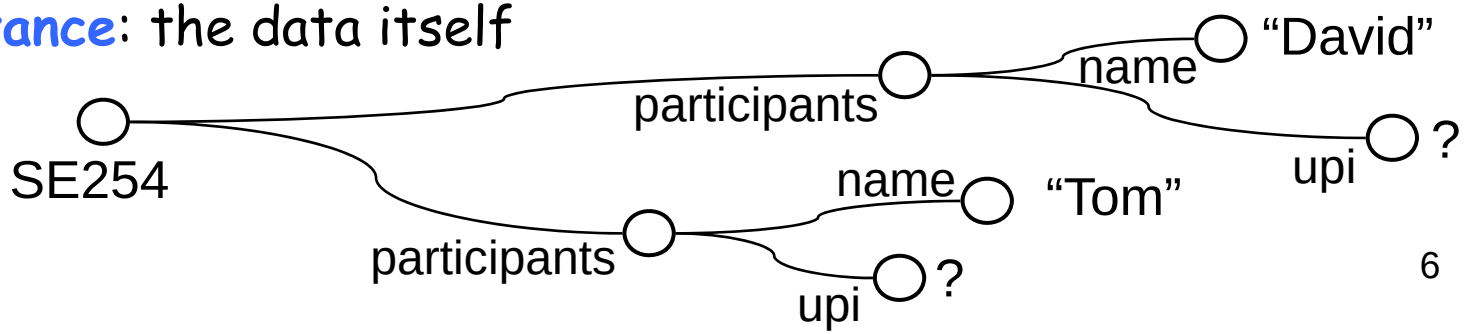
Example: the parsimonious data model (PDM)

Metamodel: describes the structure of a model

Model: describes the structure of data



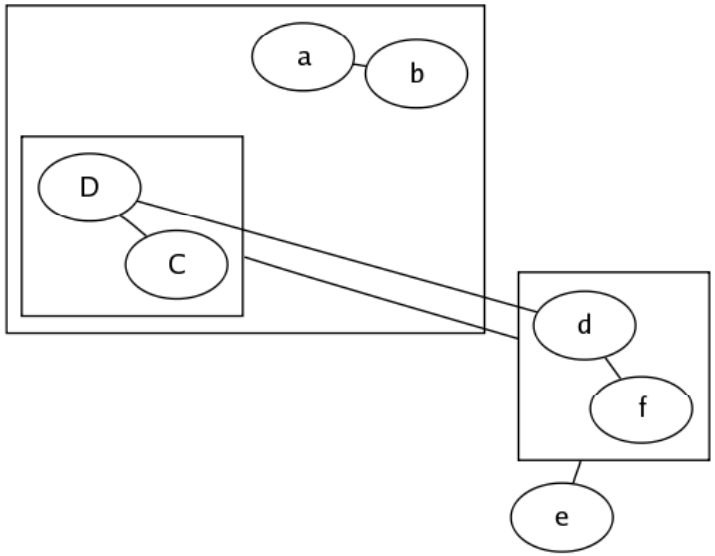
Model instance: the data itself



Domain Specific Languages (DSLs)

- Modeling is not necessarily done graphically; also textual models
- DSLs are tailored to a specific domain, i.e. they provide a model for that domain
- Much easier to describe a problem using a suitable DSL than a general purpose language like Java
- Example: GraphViz language for graphs

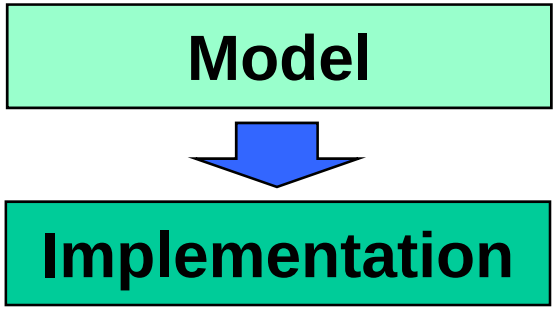
```
graph G {
    subgraph clusterA { a -- b;
    subgraph clusterC {C -- D;} }
  subgraph clusterB { d -- f }
  d -- D
  e -- clusterB
  clusterC -- clusterB
}
```



Forward and Reverse Engineering

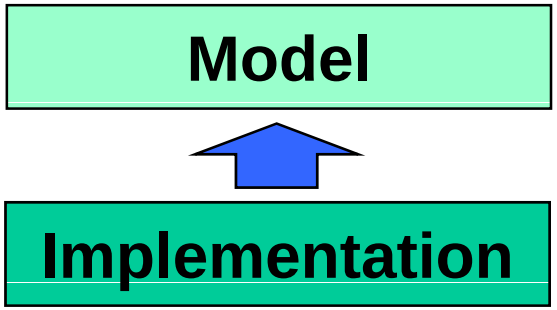
Forward Engineering

- Generate a lower-level representation of a system from a higher-level one
- Usually: generate an implementation from a model
- Examples: data model to source code, DSL to source code



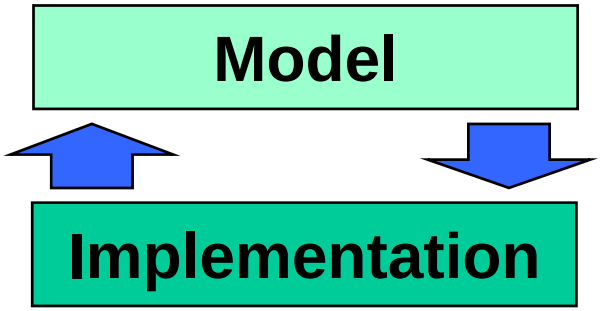
Reverse Engineering

- Recover higher-level information about a system from low-level information
- Usually: from executable implementation to model
- Examples: data model from source code, source code from a binary executable, data model from a database, documentation of legacy code



Re-Engineering and Round-Trip Engineering

Re-Engineering



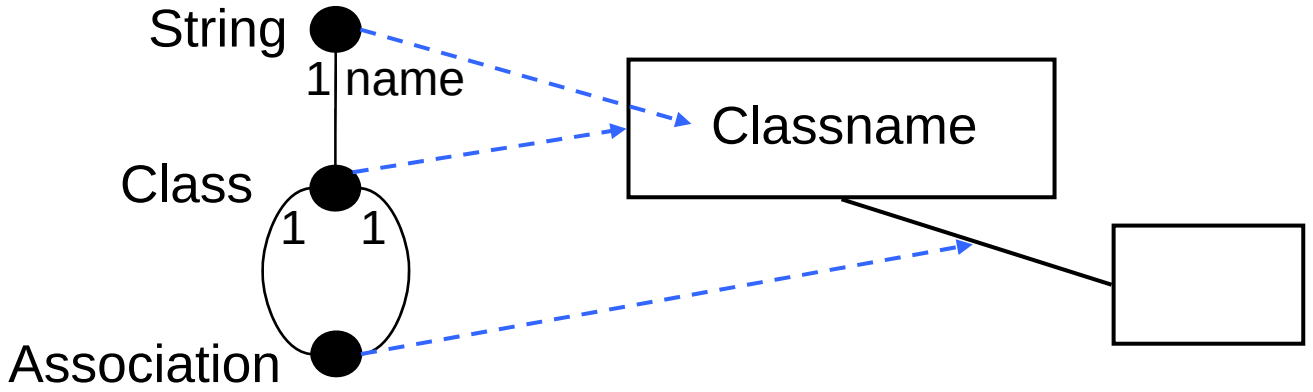
- Change an existing system by first reverse engineering information about it
- Use that information to do changes and perform forward engineering

Round-Trip Engineering

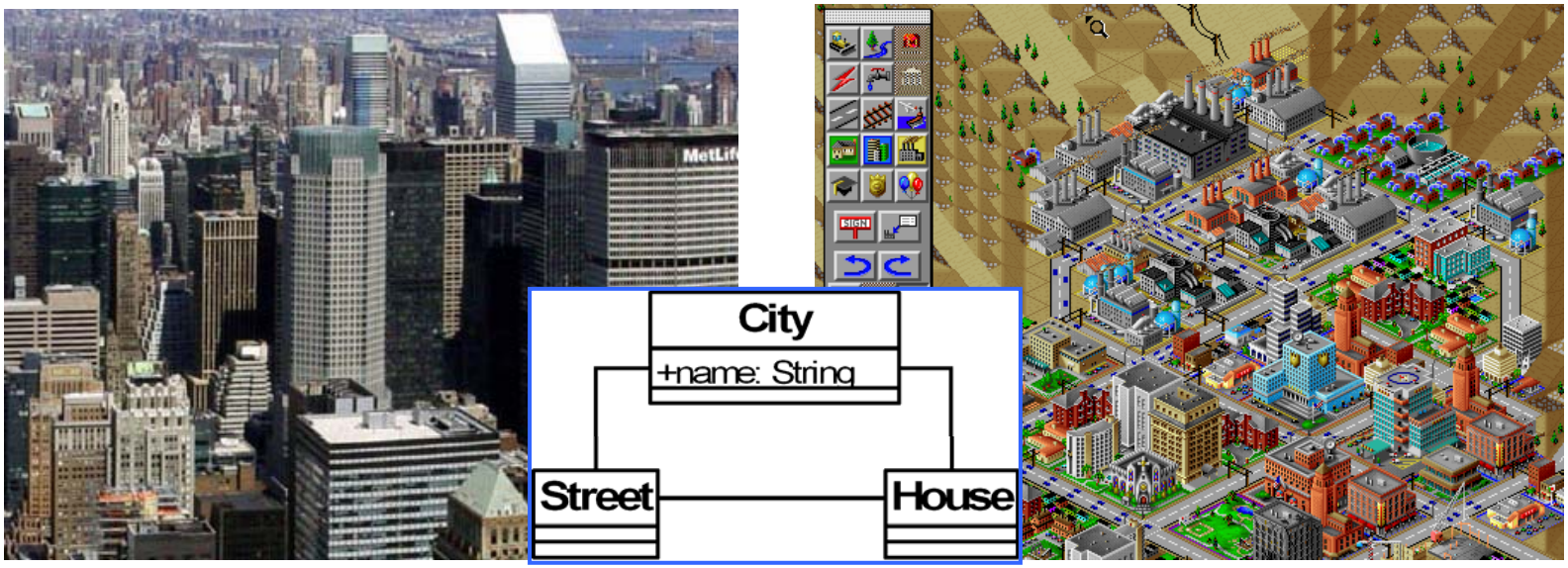
- Working with two different representations; switching between them
- Changes in one representation cause corresponding changes in the other one and vice versa (both directions)

Meta-CASE Tools

- Idea: a tool for creating CASE tools
- Many CASE modeling tools use **2D graph** for visualization, i.e. vertices and edges
- Varying shapes, labels, allowed connections, etc.
- Use commonalities for generic tool specification:
  1. Specify a data model
  2. Specify how data model elements are represented in the 2D graph
- E.g. MetaEdit+, Pounamu, Eclipse GMF



Data Modeling



Modeling an App with a 3-Tier Architecture

Presentation tier

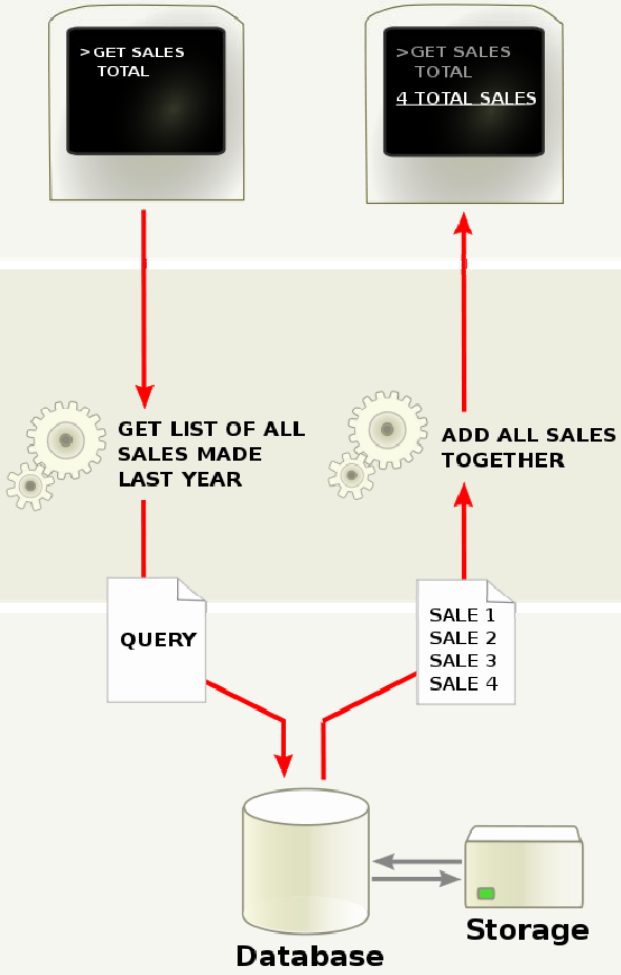
The top-most level of the application is the user interface. The main function of the interface is to translate tasks and results to something the user can understand.

Logic tier

This layer coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also moves and processes data between the two surrounding layers.

Data tier

Here information is stored and retrieved from a database or file system. The information is then passed back to the logic tier for processing, and then eventually back to the user.



- Written docs
- UI models (e.g. screen diagrams)

- Written docs
- Flow or sequence charts
- State machines

- Written docs
- Data models (e.g. class diagrams)

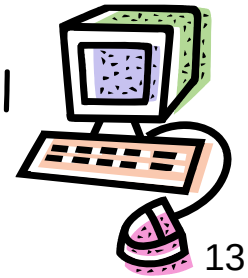
Data Modeling

- Create model of the **entities in a system** and the **relations between them** from user requirements
- Often the most important artifact (for **data-centric apps**)

Analysis data model

- What does the system deal with?
  The **concepts** as they are understood by the expert users ("domain-specific").
  -> **classes**, **attributes**, **associations**
- Allows you to **communicate** with customer & verify specification
- **No implementation** details (too early, may confuse customer)
- **Evolves** over time (too much detail too early is a waste)

Design data model extends/adapts/refines analysis model so that it becomes clear how to implement the system



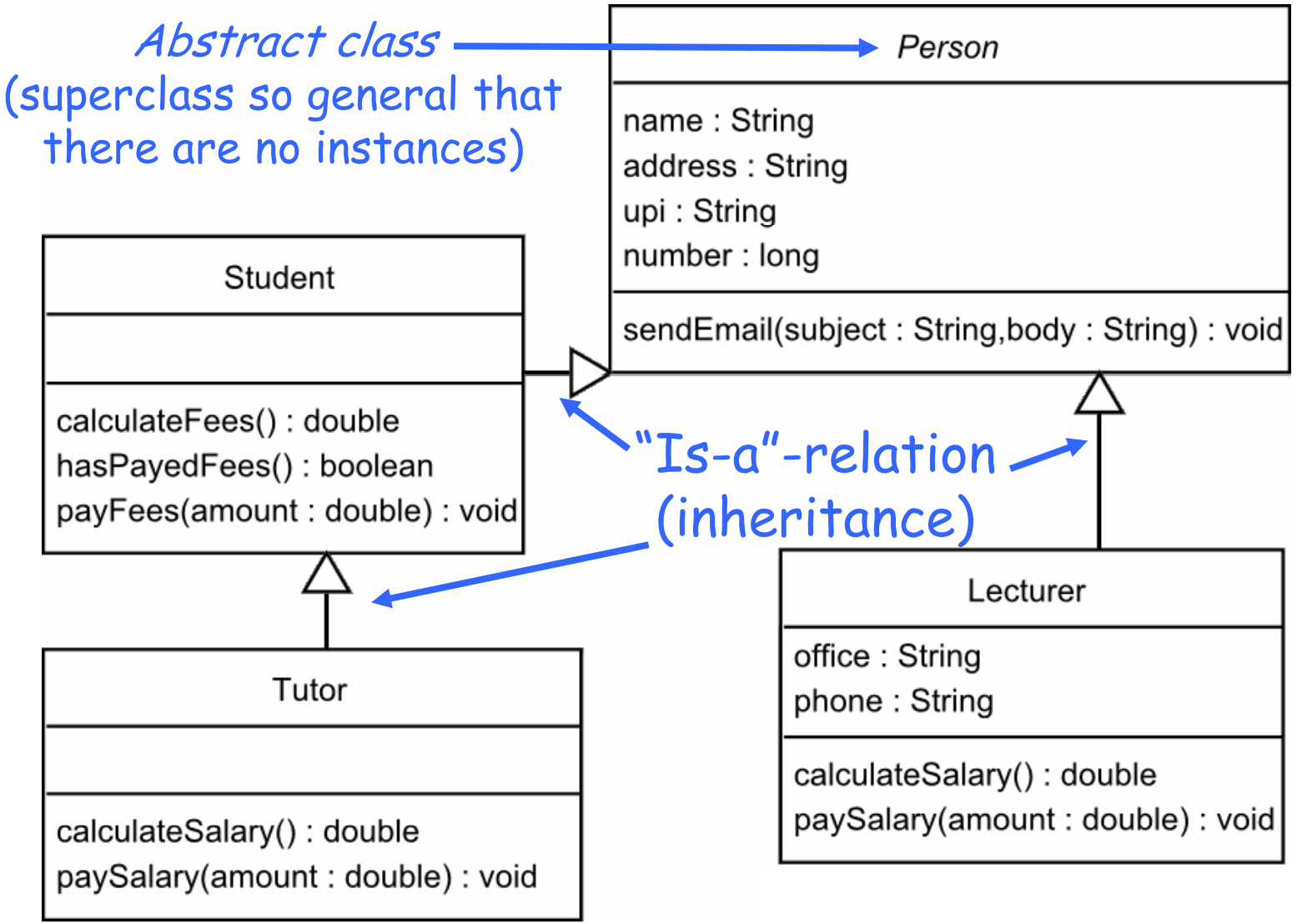
Classes

Student
Name : String Address : String upi : String number : long
calculateFees() : double hasPayedFees() : boolean payFees(amount : double) : void sendEmail(subject : String,body : String) : void

Lecturer
name : String address : String upi : String number : long office : String phone : String
calculateSalary() : double paySalary(amount : double) : void sendEmail(subject : String,body : String) : void

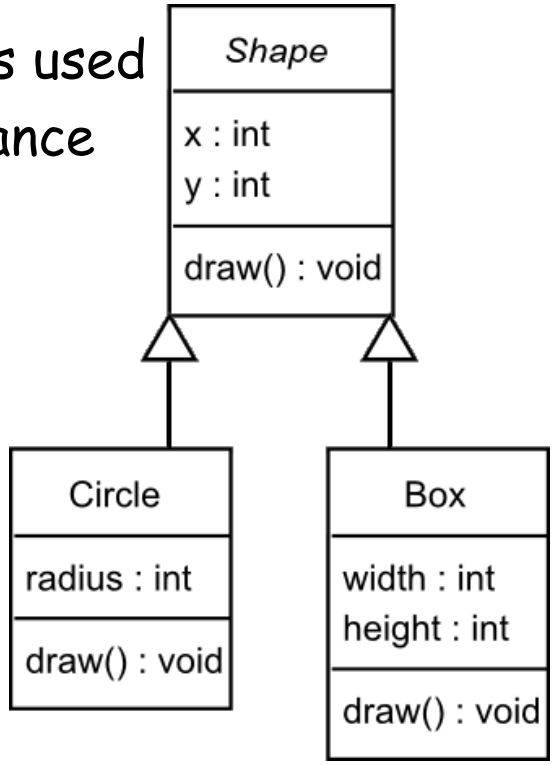
- Class name and instance variables (**attributes**)
- Possibly some method signatures (operations)
- Use operations as a reminder rather than a strict implementation decision

Reusing Classes: Inheritance / Generalization



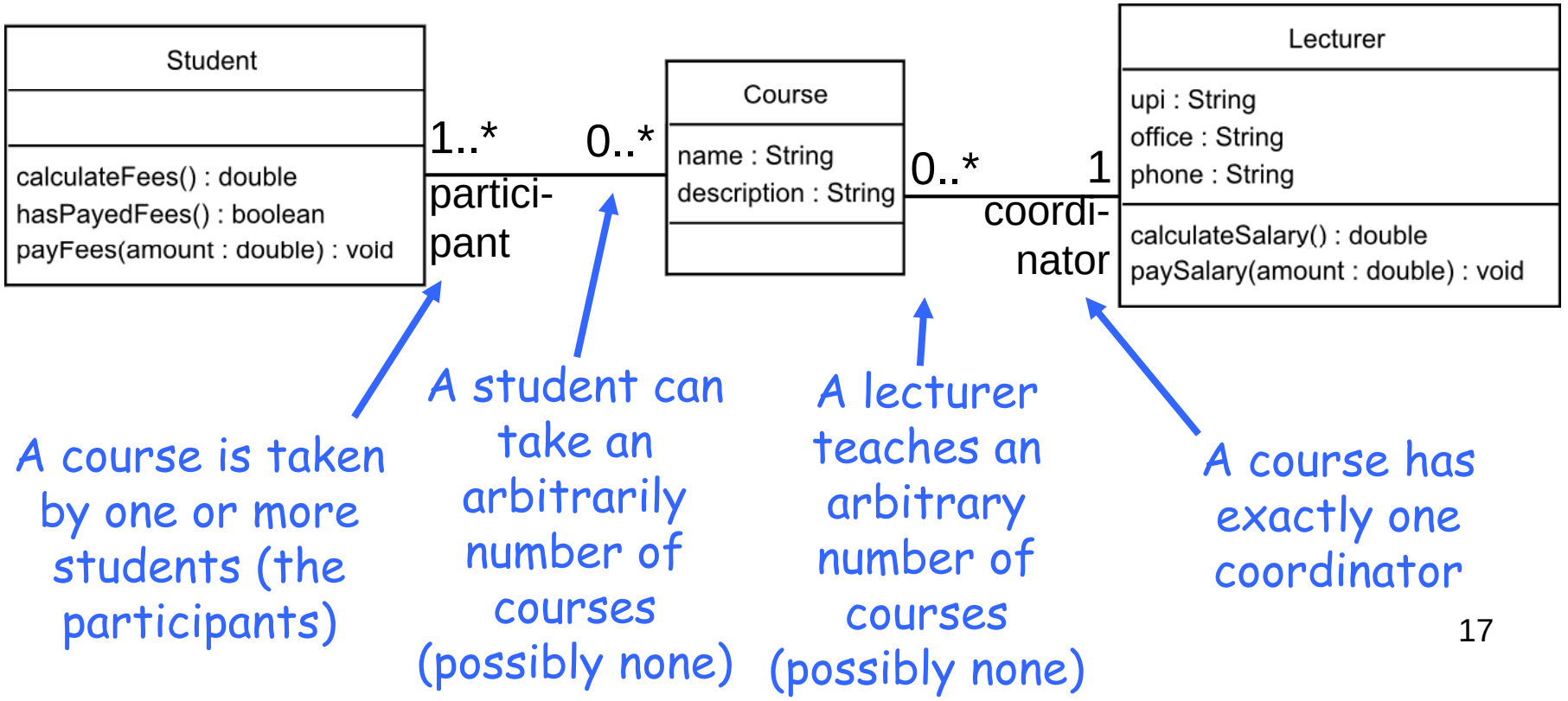
Using Inheritance in Analysis Models

- Single inheritance:
  - One general **superclass**, several specialized **subclasses**
  - **Substitution principle**:
    Subclasses can be used where superclass is used
  - Easier to understand than multiple inheritance
- Advantages:
  - Treating similar objects similarly
    ("Don't ask what kind")
  - **Reuse** of commonalities of classes
  - Better **maintenance**
- Disadvantages:
  - May be an **implementation decision**
  - May be **confusing** to the customer



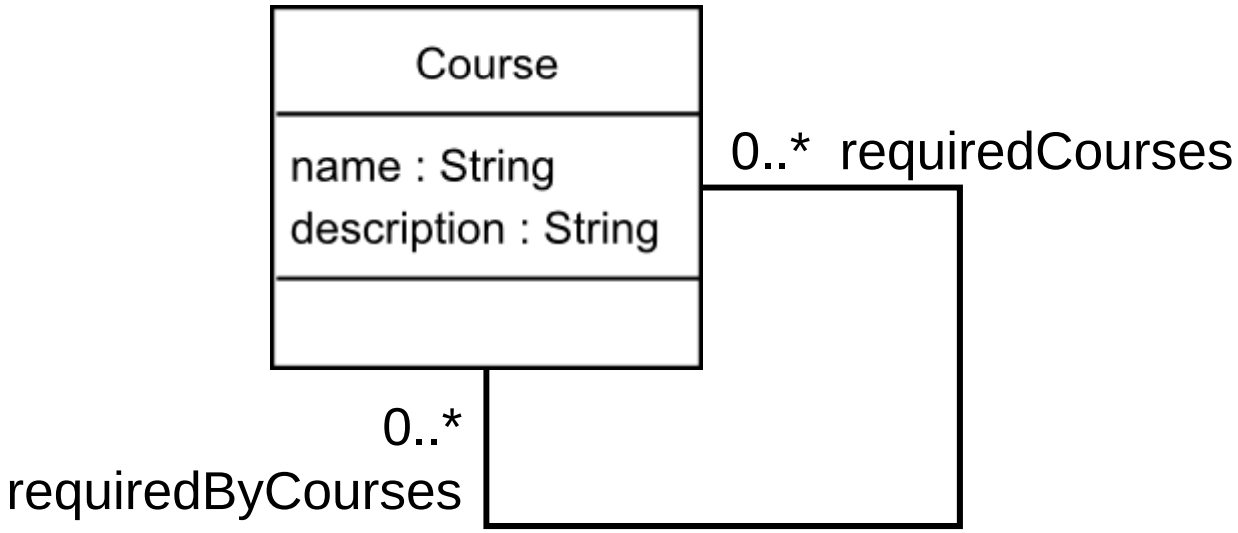
Connecting the Classes: Associations

1. **Connect classes** with a line
2. Specify **multiplicities** on each side of association: min..max (min and max of connected instances on each side)
3. Use **roles** to make the model more self-explanatory



Recursive Associations

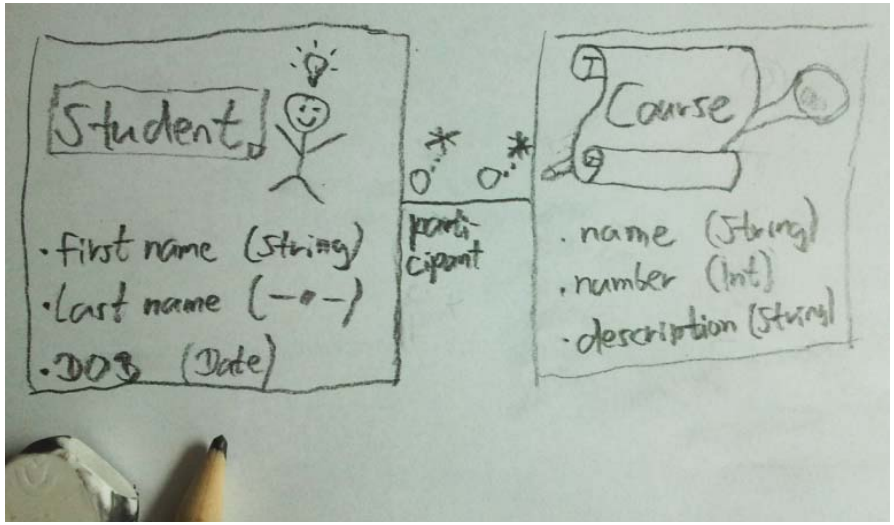
- Associations that **connect a class with itself**;
  e.g. for ordering instances (e.g. hierarchies, dependencies, flow)
- Use **roles** to distinguish ends of associations
- Example: required courses for a course
  - A course may require a student to complete 0..* other courses first
  - Each course may be required by 0..* other courses



Analysis vs. Design Model

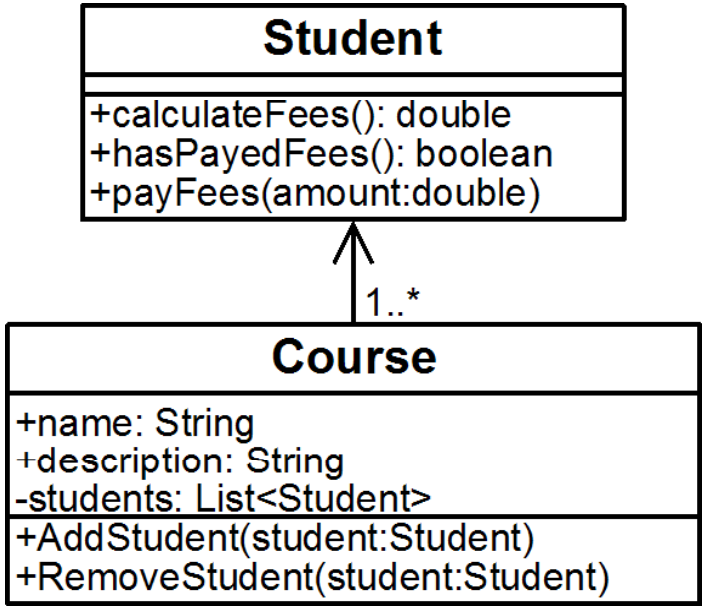
Analysis Model

- Keep it **simple** (the customer must understand it)
  e.g. only undirected associations
- Only those things that are part of the **requirements**
  (no implementation decisions)



Design Model

- Directed associations
  (Java object references)
- Access control
  (interfaces, public/private…)
- Getters and setters
- Methods for data storage, error handling, GUI, …



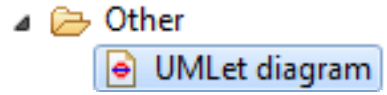
Creating Data Models with UMLet for Eclipse

Creating a Diagram

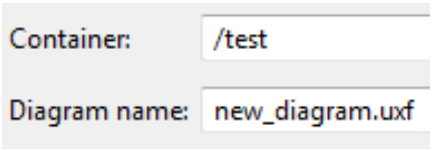
Software Engineering
The University of Auckland

First install UMLet from www.umlet.com

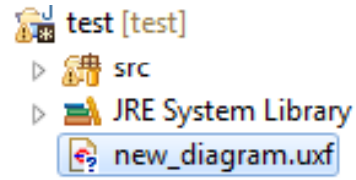
1. Select File->New->Other ->UMLet Diagram



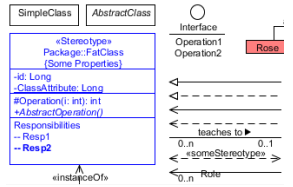
2. Choose a project ("Container") and a diagram name



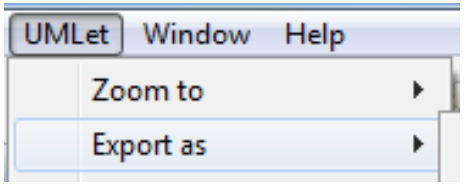
3. Open diagram file through Package Explorer



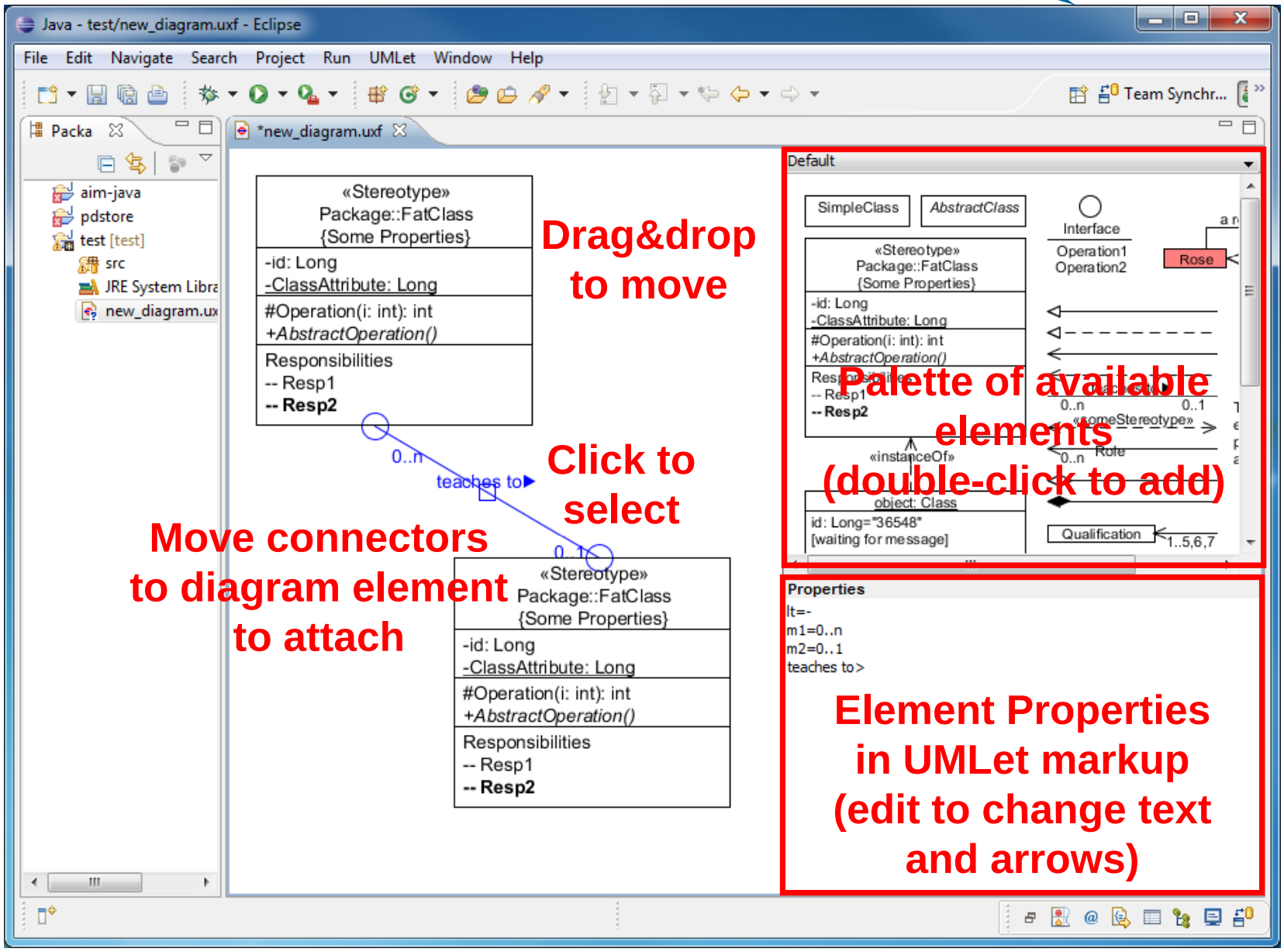
4. Add diagram parts by double-clicking palette



5. Export diagram: UMLet->Export as->PDF



Editing a Diagram



Today's Summary

- There are different levels of model data: **metamodels**, **models** and **model instances**
- **Domain Specific Languages** (DSLs) are languages for modeling particular domains
- Models are important for **forward** and **reverse engineering**, re-engineering and round-trip engineering
- **Meta-CASE tools** support the creation of graphical modeling tools

- **Analysis models** capture requirements so that the users can understand and verify them
- **Design models** refine analysis models so that it becomes clear how to implement the system
- **Modeling tools** help to create models and code

Exercise

Software Engineering
The University of Auckland

Create a class diagram for the following system:

A GP needs a software to manage her patients, appointments and invoices. The patients have a name, address and phone number, and the date on which they first visited the GP needs to be stored, too. Private patient's records have also the name of their health insurance company. A patient can have several appointments with a time and date, and several appointments can have an invoice associated with them. An appointment has at most one invoice. An invoice contains several items that have a description and a price.