Information Systems

Analysis – II

Structural Modelling

Module SITS code: COIY059H7
Dr Brian Gannon

Modelling - Recap

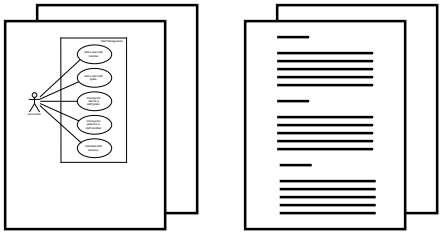
- A **system** is the overall thing that is being modelled
- A **subsystem** is a part of a system consisting of related elements
- A **model** is an abstraction of a system or subsystem from a particular perspective
- Different models present different views of the system, for example:
  - use case view
  - design view
  - process view
  - implementation view
  - deployment view

Model development

- During the life of a project using an iterative life cycle, models change along the dimensions of:
  - abstraction—they become more concrete
  - formality—they become more formally specified
  - level of detail—additional detail is added as understanding improves

Modelling – Iteration and Elaboration

Iteration 1
Obvious use cases
Simple use case descriptions



Iteration 2
Additional use cases
Simple use case descriptions
Prototypes

Iteration 3
Structured use cases
Structured use case descriptions
Prototypes/Product Increments

Structural Modelling - Introduction

- Functional models represent **system behavior**
- Structural models represent **system objects & their relationships**: People, Places, Things
- Main goal in structural modelling is to discover the **key data** contained in the problem domain and to build a structural model of the objects
- Typically requires iteration: first, business-centric (e.g. accounts, inventory), then technology-centric (databases, files)

Objects

"Objects have state, behaviour and identity."

- *State:* the condition of an object at any moment, affecting how it can behave
- *Behaviour*: what an object can do, how it can respond to events and stimuli
- *Identity*: each object is unique

Examples of Objects

object	identity	behaviour	state
a person	samuel beckett	read, write, sit	drinking, unhappy

Examples of Objects

object	identity	behaviour	state
a person	samuel beckett	read, write, sit	drinking, unhappy
a bicycle	mountain bike	move, break, stop	new, stolen, dirty

Examples of Objects

object	identity	behaviour	state
a person	samuel beckett	read, write, sit	drinking, unhappy
a bicycle	mountain bike	move, break, stop	new, stolen, dirty
a protein	haemoglobin	fold, decay, carry	oxygenated, tense, unbound

Classes

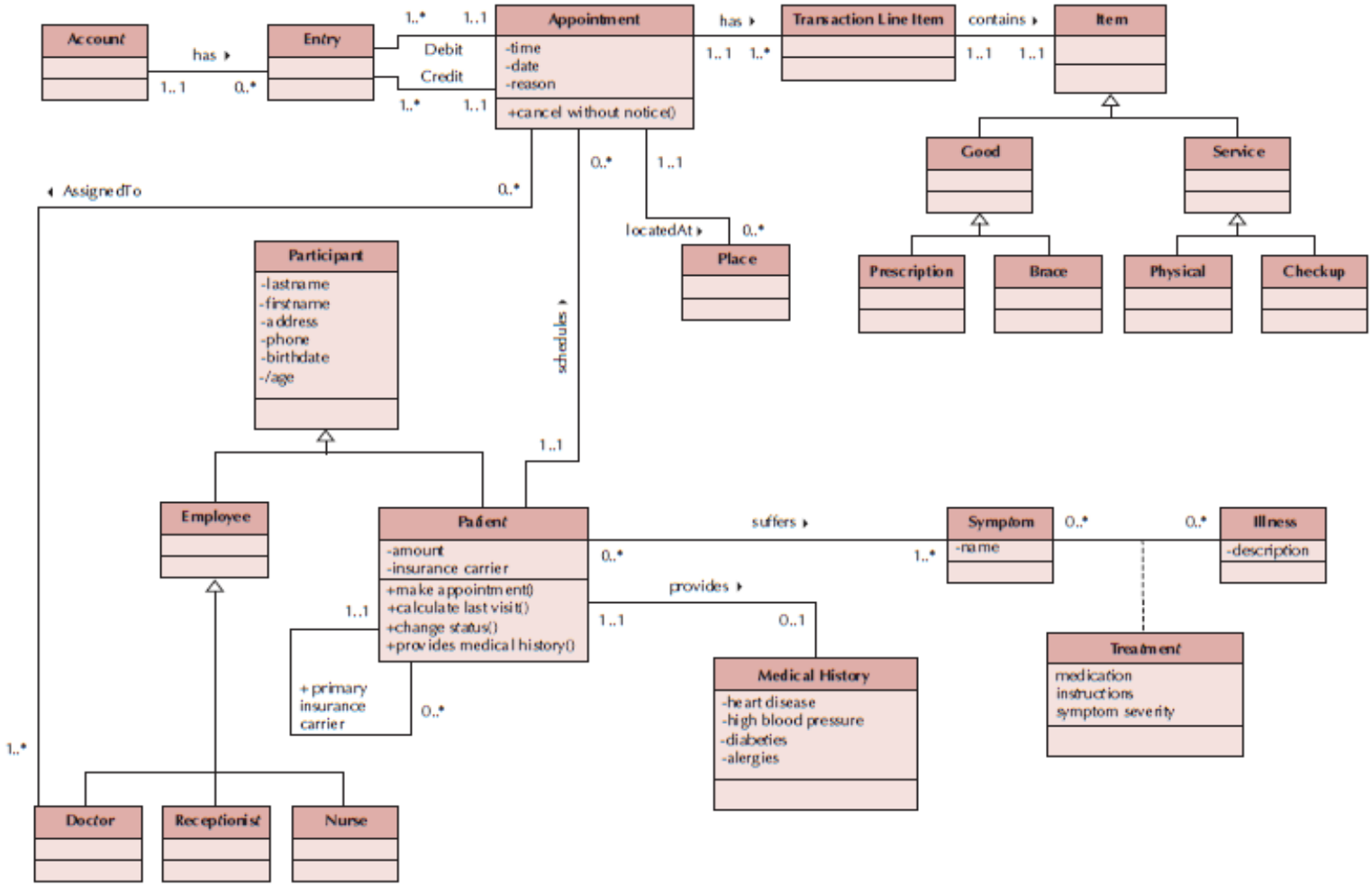
- A ***class*** is a description of a set of objects with similar features (attributes, operations); semantics; constraints

- All objects are ***instances*** of some **class** and are similar in:
  - **structure** (what they '*know*', what information they hold, what links they have to other objects)
  - **behaviour** (what they can do)

- A class diagram is a static model that shows classes and their relationships to one another

- During analysis, classes refer to the people, places, events and things about which the system will capture information

Example Class Diagram

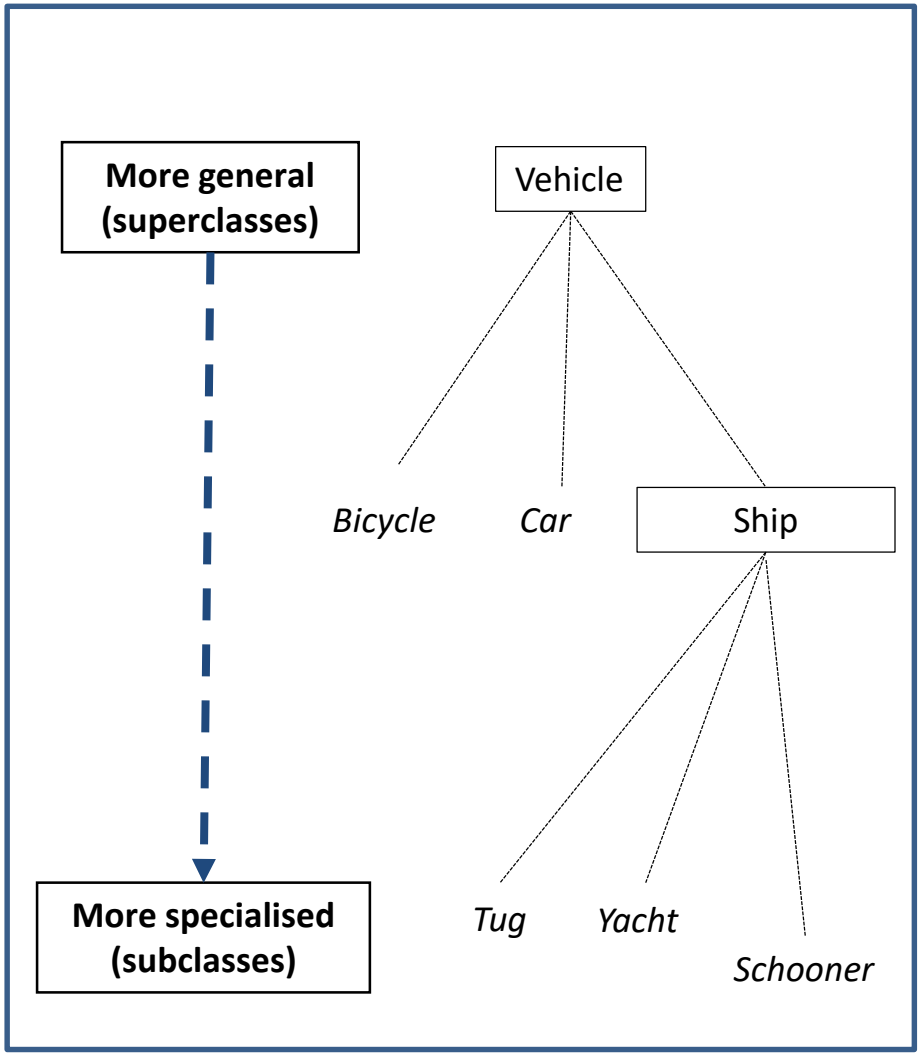


Classification and Hierarchy

- Classification is hierarchic in nature
  - a vehicle may be a bike, a car, a ship
  - a ship may be a schooner, a tug, a yacht
  - a yacht may be a yawl, a ketch, a schooner
  - …and so on

Classification and Hierarchy

- Classification is hierarchic in nature
  - a vehicle may be a bike, a car, a ship
  - a ship may be a schooner, a tug, a yacht
  - a yacht may be a yawl, a ketch, a schooner
  - …and so on



Definitions – relationships

- Generalisation
  - Represents relationships that are “a-kind-of”
  - Enables inheritance of attributes and operations

- Aggregation
  - Represents relationships that are “a-part-of” or “has-parts”
  - Relates parts to wholes or assemblies
  - Composition denotes a physical “a-part-of” relationship

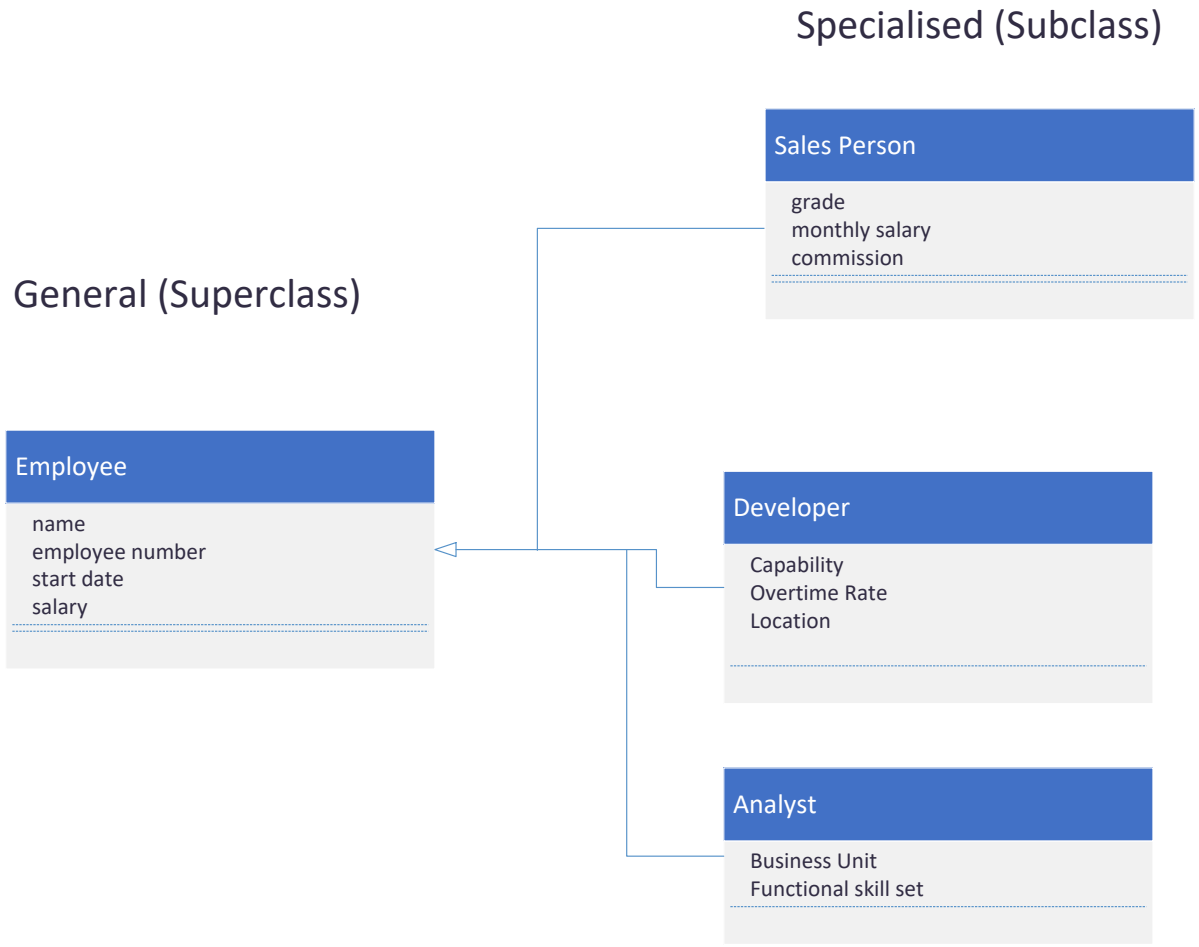
- Association
  - Represents relationships that are “linked-to” or “associated with”
  - Miscellaneous relationships between classes (usually a weaker form of aggregation)

Generalisation

- Generalisation denotes inheritance
  - Properties and operations of the superclass are valid for the sub-class
  - Depicted as a solid line with a hollow arrow pointing at the superclass

Generalisation

- Generalisation denotes inheritance
  - Properties and operations of the superclass are valid for the sub-class
  - Depicted as a solid line with a hollow arrow pointing at the superclass



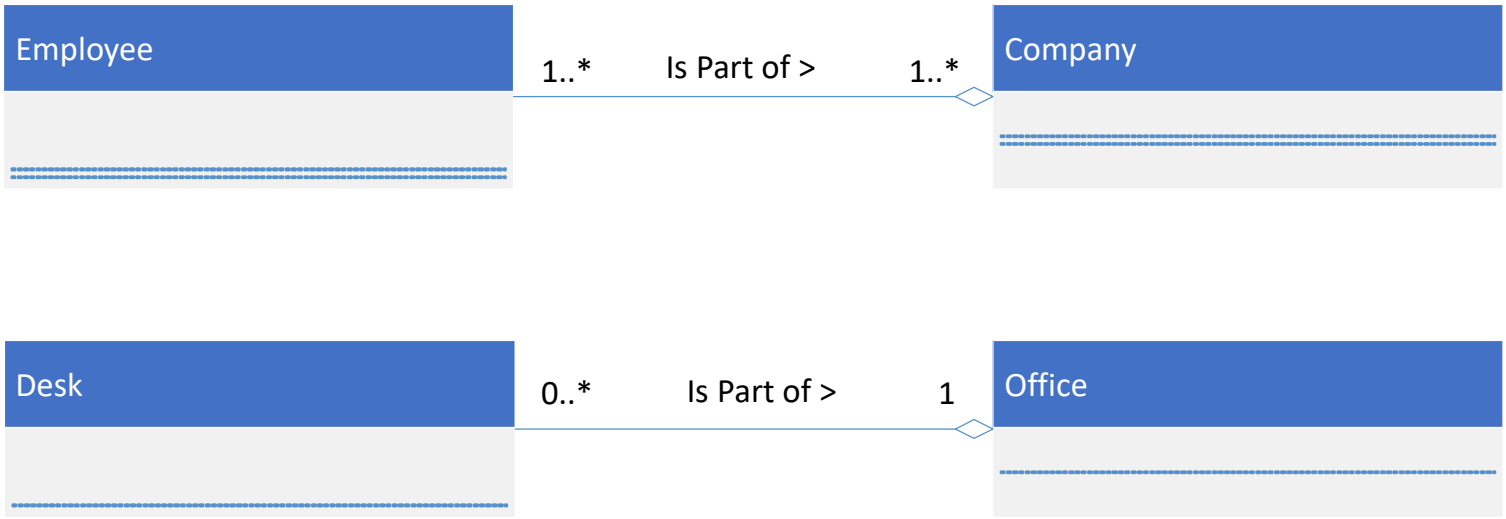
Inheritance

- The description of a superclass applies to all its subclasses, including:
  - Structure (including associations)
  - Behaviour

- Often known loosely as *inheritance*

- Inheritance is how an object-oriented programming language implements generalisation / specialisation)

Aggregation



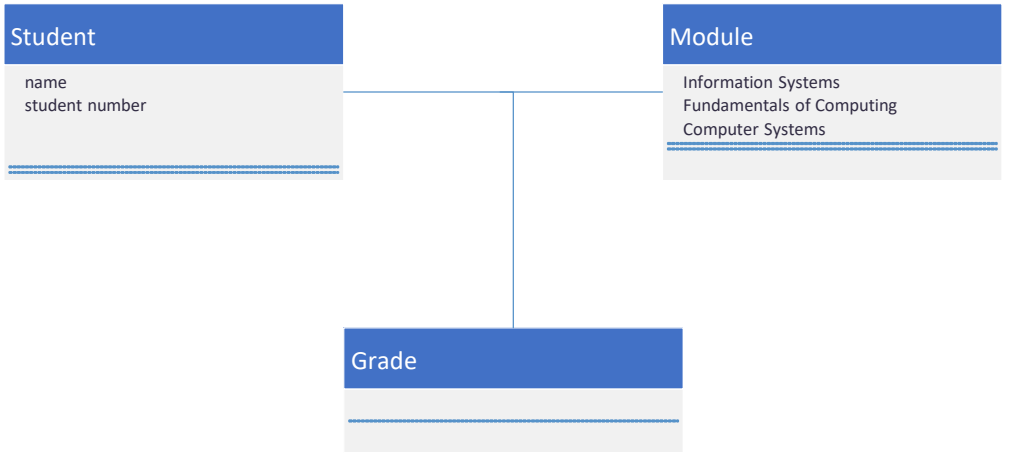
Association

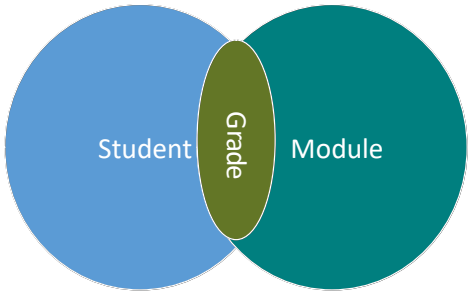
- Common in many-to-many relationships

- Used when attributes about the relationship between two classes needs to be recorded
  - Students are related to courses; a `Grade` class provides an attribute to describe this relationship
  - Illnesses are related to symptoms; a `Treatment` class provides an attribute to describe this relationship

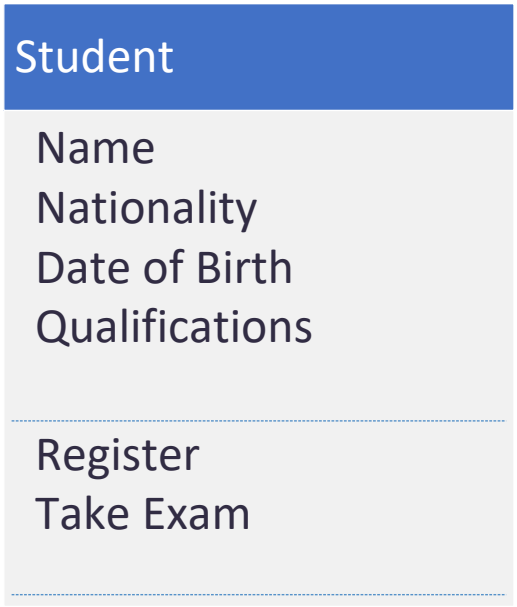
Association

- Common in many-to-many relationships
- Used when attributes about the relationship between two classes needs to be recorded
  - Students are related to courses; a `Grade` class provides an attribute to describe this relationship
  - Illnesses are related to symptoms; a `Treatment` class provides an attribute to describe this relationship

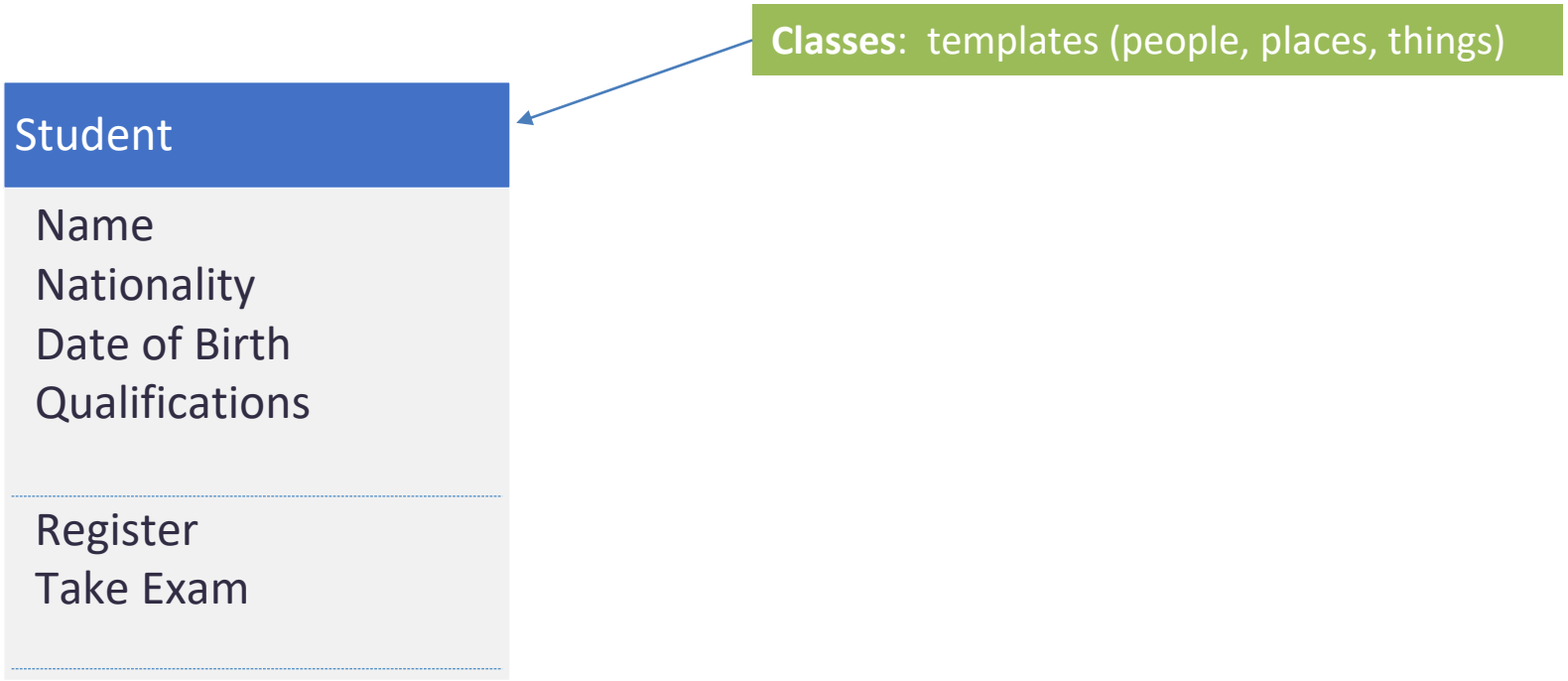




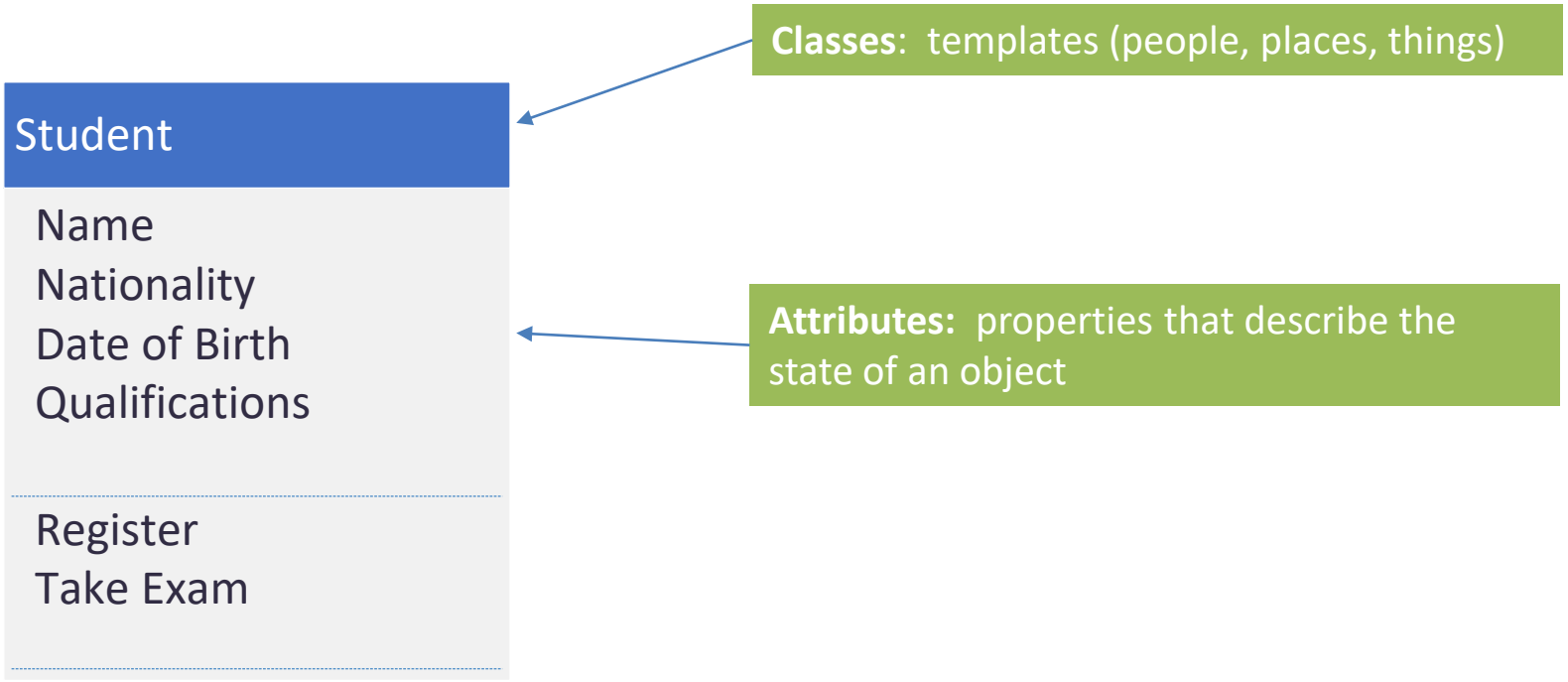
Definitions and Syntax



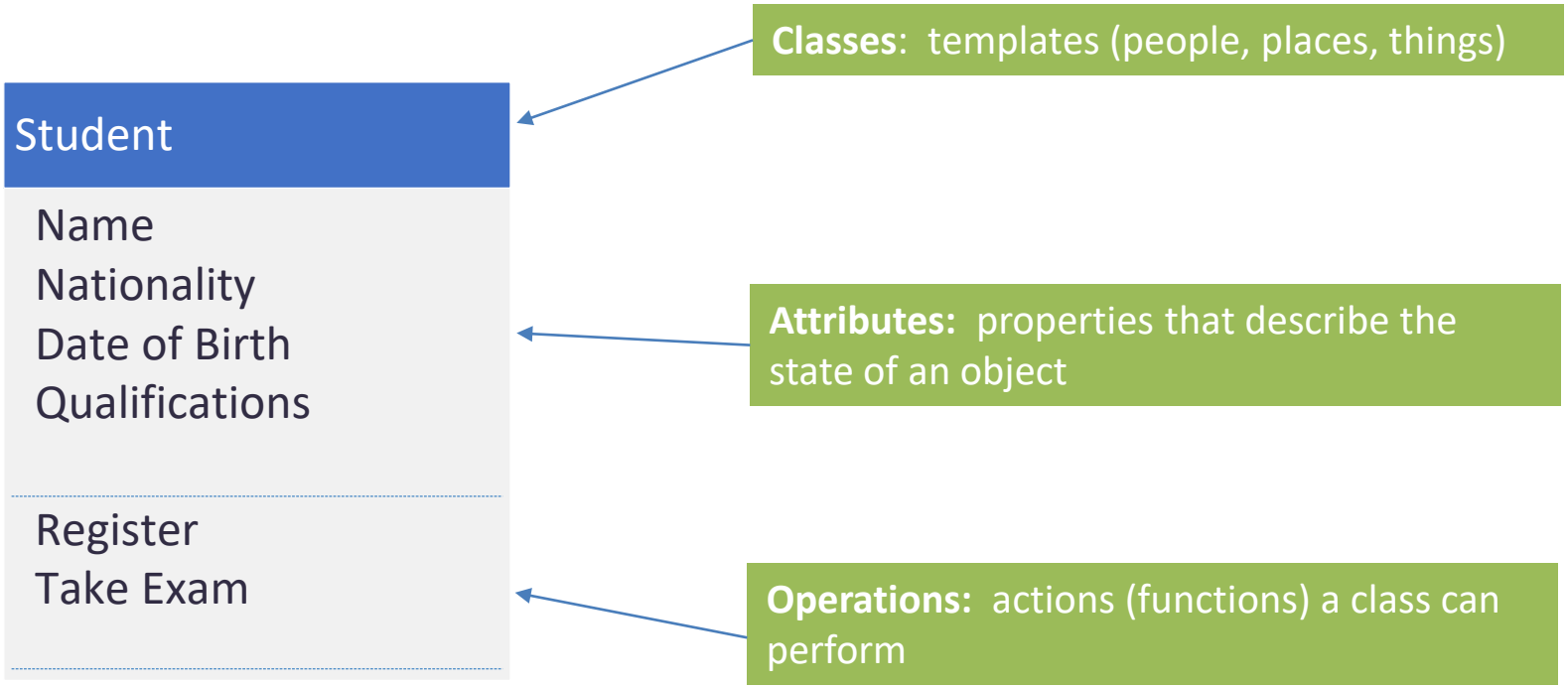
Definitions and Syntax



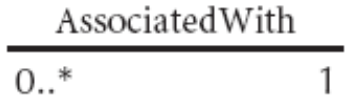
Definitions and Syntax



Definitions and Syntax



Definitions and Syntax

An association: • Represents a relationship between multiple classes or a class and itself. • Is labeled using a verb phrase or a role name, whichever better represents the relationship. • Can exist between one or more classes. • Contains multiplicity symbols, which represent the minimum and maximum times a class instance can be associated with the related class instance.	
A generalization: • Represents a-kind-of relationship between multiple classes.	
An aggregation: • Represents a logical a-part-of relationship between multiple classes or a class and itself. • Is a special form of an association.	
A composition: • Represents a physical a-part-of relationship between multiple classes or a class and itself • Is a special form of an association.	

Attributes

- Properties of a class
  - Derived attributes (/) – e.g. age is derived from date of birth
  - Public attributes (+): visible to all classes
  - Private attributes (-): visible only to an instance of the class in which they are defined
  - Protected attributes (#): visible only to an instance of the class in which they are defined and its descendants

Operations

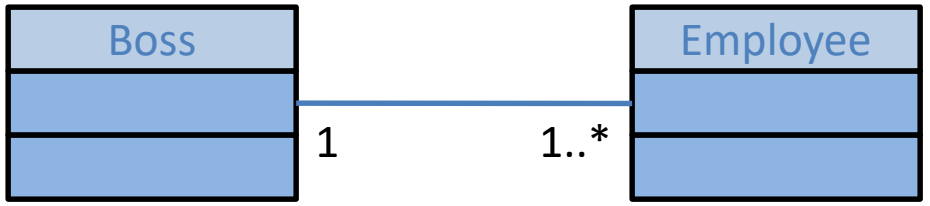
- Types of operations:
  - Constructor—creates an object
  - Query—makes information about the state of an object available
  - Update—changes values of some or all of an object's attributes
  - Destructor—deletes or removes an object

- Common operations (e.g. create/delete an instance) are not shown

Multiplicities

Exactly one:
A department has one and only one boss

Zero or more:
An employee has zero to many children



One or more:
A boss is responsible for one or more employees

Simplifying Class Diagrams

- Fully populated class diagrams of real-world system can be difficult to understand

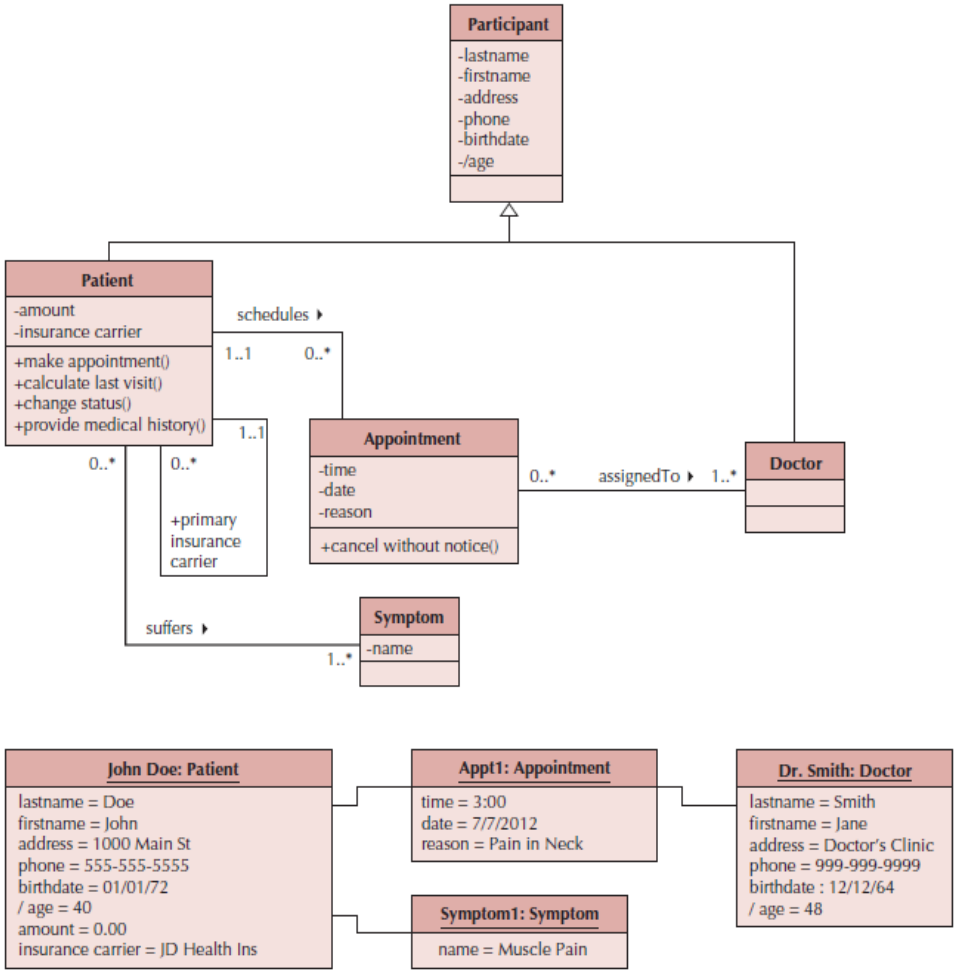
- Common ways of simplifying class diagrams:
  - Show only concrete classes
  - The view mechanism shows a subset of classes
  - Packages show aggregations of classes (or any elements in UML)

Object Diagrams

- Class diagrams with instantiated (concrete) classes
- Used to discover additional attributes, relationships and/or operations or those that are misplaced

Object Diagrams

- Class diagrams with instantiated (concrete) classes
- Used to discover additional attributes, relationships and/or operations or those that are misplaced



Identifying Objects

- Textual analysis of use case information
  - Nouns suggest classes
  - Verbs suggest operations
  - Creates a rough first cut to provide an object list

- Common Object Lists
  - Physical things
  - Incidents
  - Roles
  - Interactions

- Brainstorming—people offering ideas
  - Initial list of classes (objects) is developed
  - Attributes, operations and relationships to other classes can be assigned in a second round

Identifying Classes

- Use activity diagrams / main use cases to identify actors and objects

- Class–Responsibility–Collaboration (CRC)
  - a technique used to help discover objects, attributes, relationships, operations
  - scenario planning by team members - what can I do? and what do I know?
  - scenario roles based on actors and objects
  - team members perform each step in the scenario
  - discover and fix problems until a successful conclusion is reached
  - repeat for remaining use-cases

CRC Cards

Class Name: Old Patient	**ID:** 3	**Type:** Concrete, Domain
Description: An individual that needs to receive or has received medical attention		**Associated Use Cases:** 2

Responsibilities	**Collaborators**
Make appointment	Appointment
Calculate last visit	
Change status	
Provide medical history	Medical history

CRC Cards

Attributes:

Amount (double)

Insurance carrier (text)

Relationships:

Generalization (a-kind-of): Person

Aggregation (has-parts): Medical History

Other Associations: Appointment

Reasonability Checks for Candidate Classes

- A number of tests help to check whether a candidate class is reasonable
  - Is it beyond the scope of the system?
  - Does it refer to the system as a whole?
  - Does it duplicate another class?
  - Is it too vague?
  - Is it too tied up with physical inputs and outputs?
  - Is it really an attribute?
  - Is it really an operation?
  - Is it really an association?

- If any answer is ‘Yes’, consider modelling the potential class in some other way (or do not model it at all)

Developing Structural Models

1. Review Use Cases
2. Identify main actors & objects
3. Identify missing objects, attributes, operations and/or relationships
4. Role-play the CRC cards—look for breakdowns & correct; create new cards as necessary
5. Create a draft class diagram
6. Review the class diagram—remove unnecessary classes, attributes, operations and/or relationships

Verifying and Validating Structural Models

- Analyst presents to developers & users
  - Walks through the model
  - Provides explanations & reasoning behind each class
- Attributes each have a data type (e.g. salary implies a number format)
- Relationships must be properly depicted on the class diagram
  - Aggregation/Association
  - Multiplicity
- Association classes are used only to include attributes that describe a relationship

Use Case Realisation

- Requirements (use cases) are usually expressed in user language
- Use cases are units of development, but they are not structured like software
- The software we will implement consists of **classes**
- We need a way to translate requirements into classes
- The ultimate product of use case realisation is the software implementation of that use case

Sample java code for implementation of a `Bicycle` class

```
public class Bicycle {

   // the Bicycle class has
   // three fields
   public int cadence;
   public int gear;
   public int speed;

   // the Bicycle class has
  // one constructor
   public Bicycle(int startCadence, int startSpeed, int startGear) {
      gear = startGear;
      cadence = startCadence;
      speed = startSpeed;
   }

   // the Bicycle class has
  // four methods
   public void setCadence(int newValue) {
      cadence = newValue;
   }

   public void setGear(int newValue) {
      gear = newValue;
   }

   public void applyBrake(int decrement) {
      speed -= decrement;
   }

   public void speedUp(int increment) {
      speed += increment;
   }

}
```

Sample class declaration for a `MountainBike` class

```
public class MountainBike extends Bicycle {

   // the MountainBike subclass has
   // one field
   public int seatHeight;

   // the MountainBike subclass has
   // one constructor
   public MountainBike(int startHeight, int startCadence,
                       int startSpeed, int startGear) {
      super(startCadence, startSpeed, startGear);
      seatHeight = startHeight;
   }

   // the MountainBike subclass has
   // one method
   public void setHeight(int newValue) {
      seatHeight = newValue;
   }

}
```

`MountainBike` is a subclass of Bicycle.

`MountainBike` inherits all the fields and methods of `Bicycle` and adds the field `seatHeight` and a **method** to set it (mountain bikes have seats that can be moved up and down as the terrain demands).

References

Dr. BRIAN GANNON

E: b.gannon@bbk.ac.uk
E: brian.gannon@vesime.com
M: 07870 153855
LI: http://lnkd.in/QDXaH6
Tw: @bjgann