Object Oriented Analysis and Design (OOAD)

COMP H2031

Lecturer: Frances Murphy

Module Aims

- **An appreciation and understanding of the software development life-cycle within the paradigm of object-orientation**
- **Ability to read and interpret the fundamental artefacts of the UML (Unified Modelling Language)**
- **Ability to carry out functional, behavioural, and data modelling using a CASE (Computer Aided Software Engineering) tool**
- **Ability to apply the UML throughout the entire software development lifecycle from requirements gathering through to deployment and delivery**
- **Be able to design reusable and pluggable object-oriented software.**

Module Assessment – 2014

January 2014

- Written Exam: 50% (see past papers)
- Assessments: 50%
  - Weekly Lab exercises: 10%
  - Two in-class tests: 20% each

Repeat Exams: August 2014

- Written Exam: 100%

- Plagiarism is not tolerated, copied work will receive 0 marks, ‘donators’ of work will also be severely dealt with

Recommended Texts

Recommended Text

- Bennett, Simon et al. (2002), Object-Oriented Systems Analysis and Design using UML, 2nd Edition, Addison-Wesley.
- Fowler, Scott (2004). UML Distilled. 3rd Edition. Addison-Wesley
- L Maciaszek (2007) Requirements Analysis & System Design. 3rd Edition. Addison-Wesley

Supplementary Materials

- Lots of material on UML, on Internet
- Reed, Paul Jr. (2002), Developing Applications with Java and UML, Addison-Wesley.
- Cockburn Alistair et al. (2001), Writing Effective Use Cases, Addison-Wesley.

UML Software

- Rational Rose (in ITB)
- Eclipse (in ITB)

Shareware Software

- Visual Paradigm for UML
- UMLet
- Etc.
- Good examples of UML diagrams provided

1 Introduction to OOAD

1.1 Overview and Schedule

1.2 Systems Development Life Cycle (SDLC)

1.3 What is OOAD?

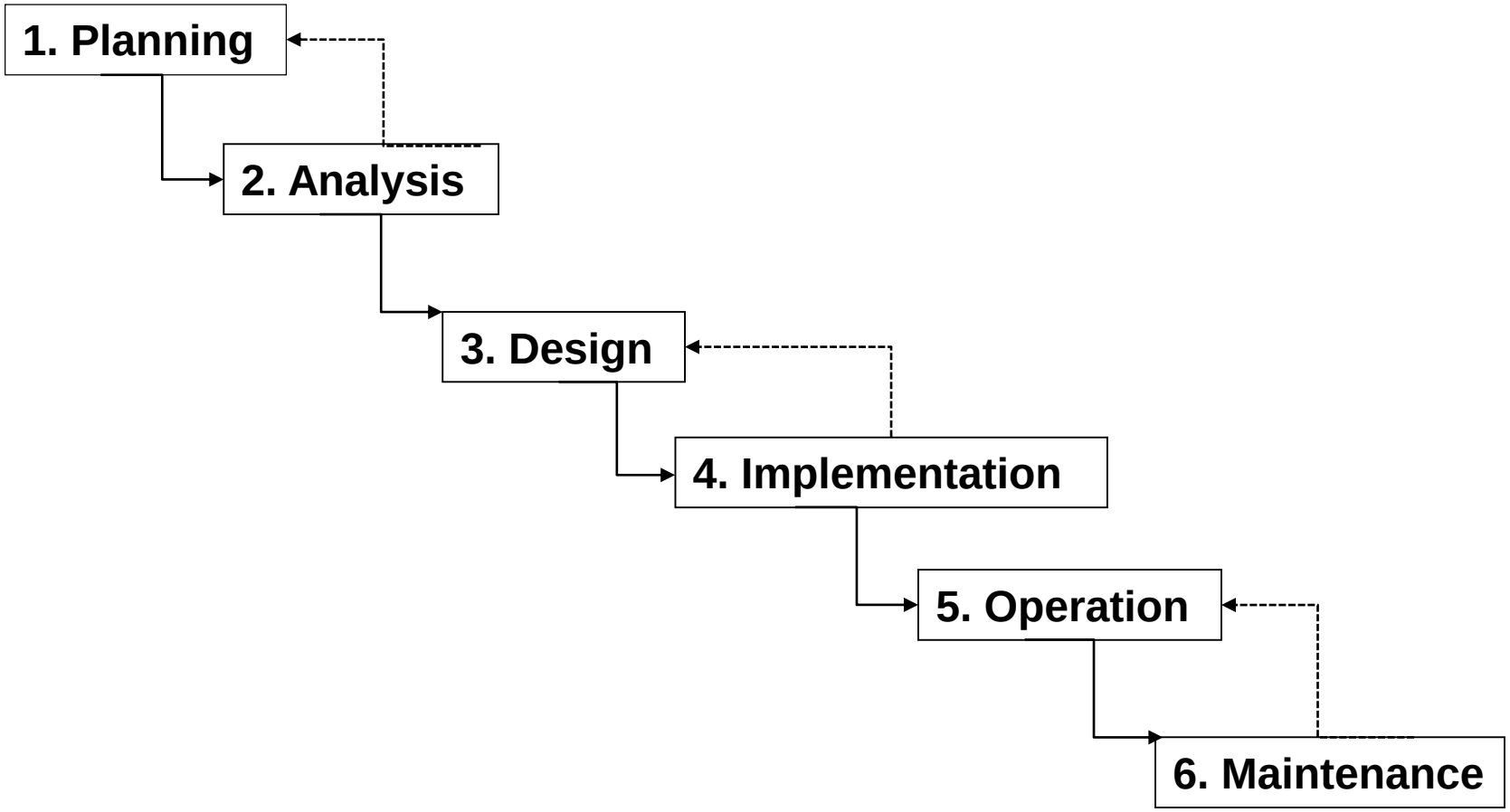
1.4 Why OOAD?

1.5 Complex Systems

1.6 The Object Model

1.1 Lecture Schedule (Provisional)		
Lecture	**Date**	**Content**
1	19 Sept	SDLC, Intro OOAD, UML
2	26 Sept	Use Case
3	3 Oct	Use Case & Sequence Diagram
4	10 Oct	Class Diagram
5	17 Oct	Classes/Objects/Constructors (Java)
6	24 Oct	Inheritance & Polymorphism (Java)
7	7 Nov	Aggregation (Java)
		In-class Test 1
8	14 Nov	Activity Diagram
9	21 Nov	State Chart Diagram
10	28 Nov	Collaboration & Package Diagram
11	5 Dec	RUP
12	12 Dec	In-class Test 2
13	19 Dec	Revision

1.2 Systems Development Life Cycle (SDLC) Phases



1.2 Systems Development Life Cycle (SDLC)

1. **Planning:**

- Understand why an information system should be built and
- Determine how the project team will go about building it.

2. **Analysis:**

- Answer the questions: who will use the system, what the system will do, and where and when it will be used.
- Investigate any current system(s), identifies improvement opportunities, and develops a concept for the new system.

1. 2 Systems Development Life Cycle (SDLC)

3. <u>Design</u>:

- Decide how the system will operate, in terms of:
  - hardware, software, and network infrastructure;
  - user interface, forms and reports; and
  - specific programs, databases, and files that will be needed.

4. <u>Implementation</u>:

- System is actually built (or purchased, in the case of a packaged software design).
- This is the phase that usually gets the most attention, because for most systems it is longest and most expensive single part of the development process.

1.2 Systems Development Life Cycle (SDLC)

5. <u>Operation</u>

- Change over from the existing business solution, whether it is software or not

6. <u>Maintenance</u>

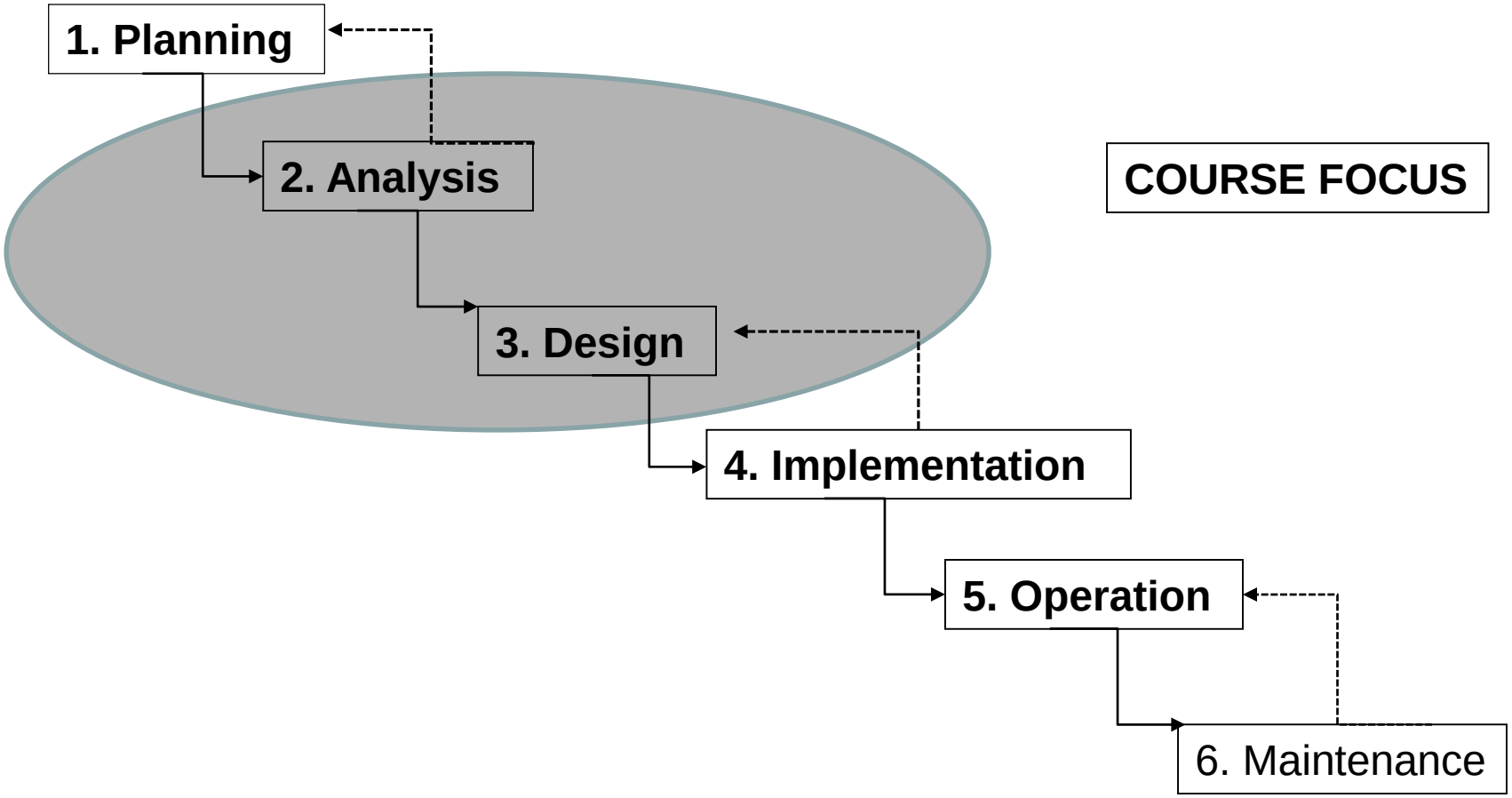
- Longest part of software life cycle

i. Housekeeping Maintenance

ii. Perfective maintenance

iii. Adaptive maintenance

1.2 Systems Development Life Cycle (SDLC)



1.3 What is OOAD?

- A method for designing and building large programs with a long lifetime
  - Blueprints of systems developed before coding
  - Iterative development process
  - Maintenance and modifications
  - Control of dependencies
  - Separation into components

Explain

- A Blueprint is ……
- Iterative means ……
- Maintenance Phase is when …….
- ……
- Modification ……
- Separating into components means ……..

1.4 Why OOAD?

- Object-orientation is closer to the way problems appear in life
- These problems generally don't come formulated in a procedural manner
- We think in terms of "objects" or concepts and relations between those concepts
- Modelling is simplified with OO because we have objects and relations

OO Approach

- Use classes, objects, methods, and messages.
- The object is a basic unit that operates in the object-oriented world.
- Objects are created from classes which act as templates
- Objects contain the ability to take action in the form of methods
- Objects send and receive messages to and from other objects.

OOAD as a Modeling Technique

- **We perceive the world as a world of objects.** Look around and you see a chair, a table, a person, and the objects are related in one way or another.

- The object-oriented technique tries to **imitate the way we think**
- A system developed by using the OO technique can be seen as a system consisting of a number of objects which cooperate to solve a task.

1.4 Why OOAD?

- OOAD is a software engineering practice
  - manage large projects professionally
  - Software becomes more complex as program size increases
- Need for tools to deal with complexity **=>** OOAD provides these tools
- Tool: CASE Tools e.g. Rational Rose
- Shareware tools e.g.

1.5 Complex Systems

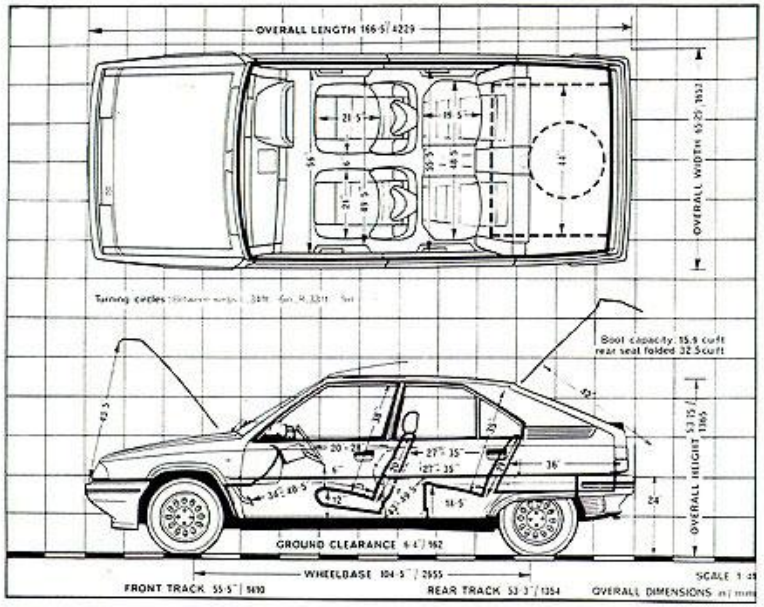
- For our purpose complex systems (Booch):
  - have many states, i.e. large "phase space",
  - are hard to comprehend in total
  - hard to predict
- Examples:
  - ant colony, an ant
  - Computer program
  - weather
  - a car

1.5 Complex Systems

- Attributes of complex systems
  - Hierarchical
  - Components
  - Primitive components
  - Few kinds of subsystems in many different combinations
  - Evolved from a simpler system

1.5 Hierarchical

- Composed of interrelated subsystems
  - subsystems consist of subsystems too
  - until elementary component

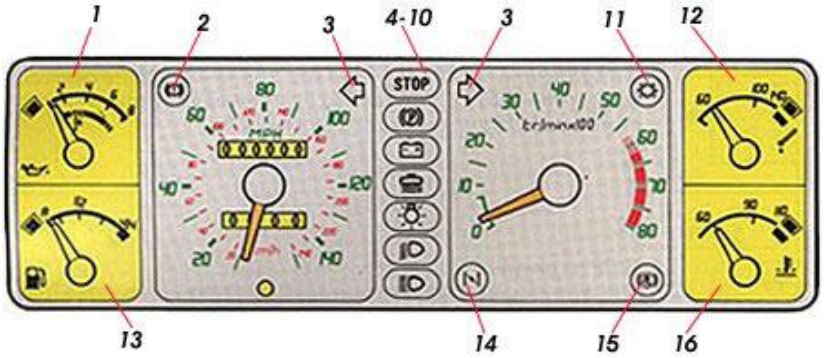


1.5 Components

- Links (dependencies) within a component are stronger than between components
  - inner workings of components separated from interaction between components
  - service/repair/replace components

1.5 Primitive Components

- There are primitive components
  - but defintion of primitive may vary
  - Nuts, bolts, individual parts?
  - replaceable components?



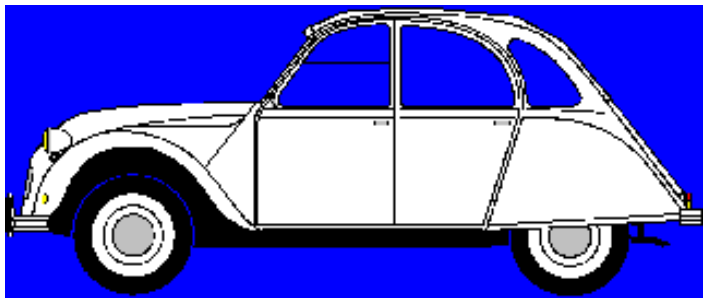
Instrument panel
or screws, bulbs and parts?

1.5 Few kinds of subsystems in many combinations

- There are common patterns
  - Nuts, bolts, screws interchangeable
  - Cables, bulbs, plugs
  - Belts, chains
  - Hoses, clamps

1.5 Evolved from a simpler system

- Complex system designed from scratch rarely works
- Add new functionality/improvements in small steps



1.5 Complex Systems: Analysis

- Have we seen it before?
- Have we seen its components before?
- Decompose by functionality ("part of")
  - Engine, brakes, wheels, lights
- Decompose by component classes ("is a")
  - The BX A8A Turbo diesel is an engine
  - Lockheed disk brake is a brake
  - ladybird is a

1.5 Complex Systems: Summary

- Have "large phase space"
- Hard to predict behaviour
- Five properties:
  - Hierarchies, Components, Primitives, Not too many kinds of components, Evolved

1.6 The Object Model

- Four essential properties (Booch)

1. Abstraction
2. Encapsulation
3. Modularity
4. Hierarchy

1.6 Abstraction

The characteristics of an object which make it unique and reflect an important concept

1.6 Encapsulation

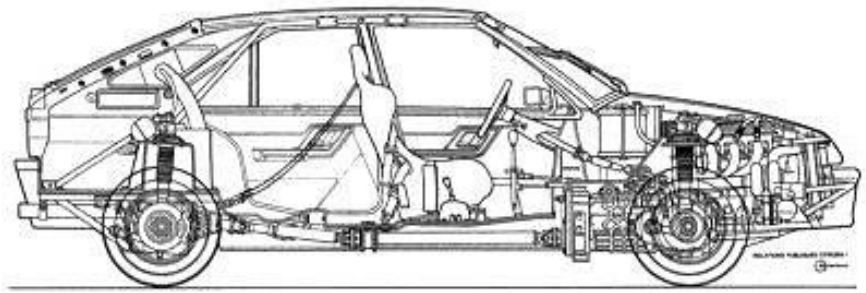
Separates interface of an abstraction from its implementation

Abstraction:	car
Interface:	steering, pedals, controls
Implementation:	you don't need to know, quite different between different makes or models

1.6 Modularity

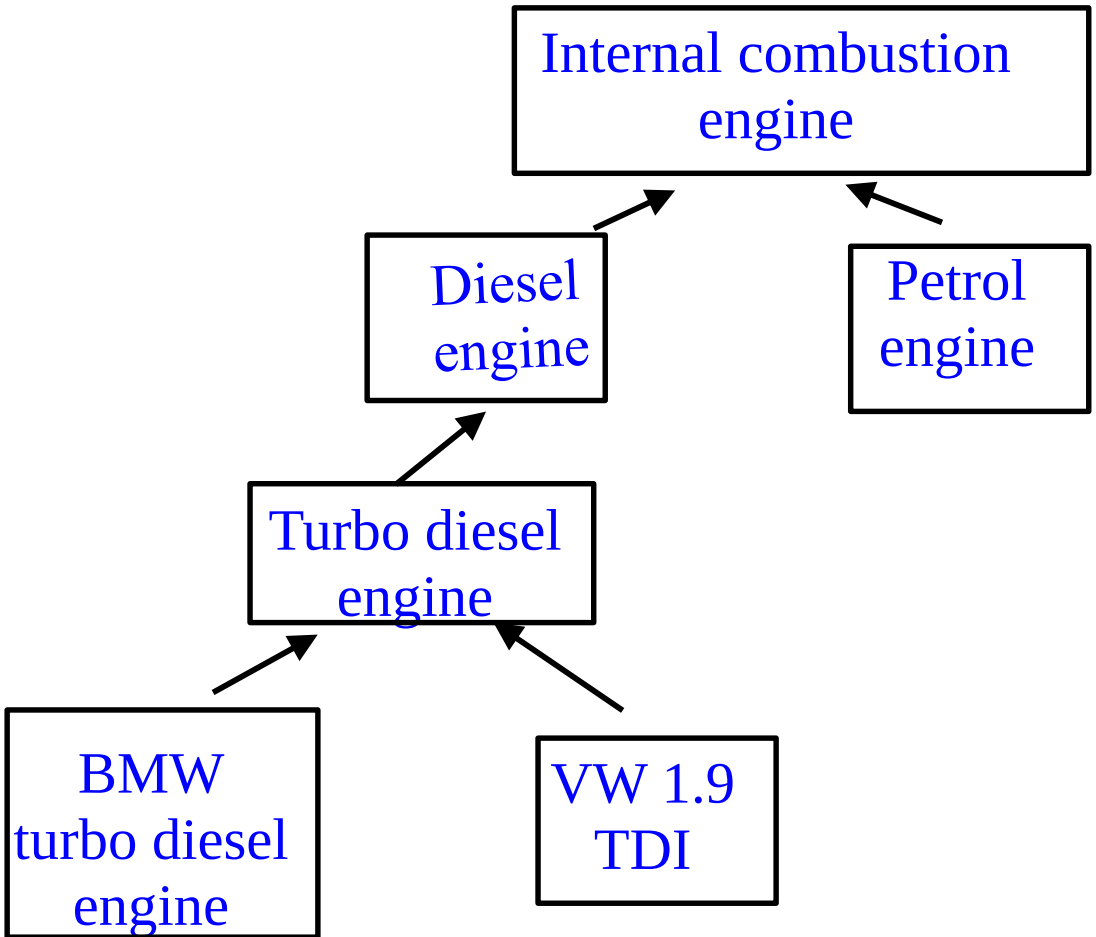
Property of a system decomposed into cohesive and loosely coupled modules

Cohesive:	group logically related abstractions
Loosely coupled:	minimise dependencies between modules



1.6 Hierarchy

Hierarchy is a ranking or ordering of abstractions



Let’s apply this to OO Software Development

Object-Orientation is a Modelling Technique

- We perceive the world as a world of objects.
- Typically many ways to classify objects

Jim might be classified as a

- **person,**
- **a male person,**
- **an employee,**
- **a student**
- **Etc..... the context decides**

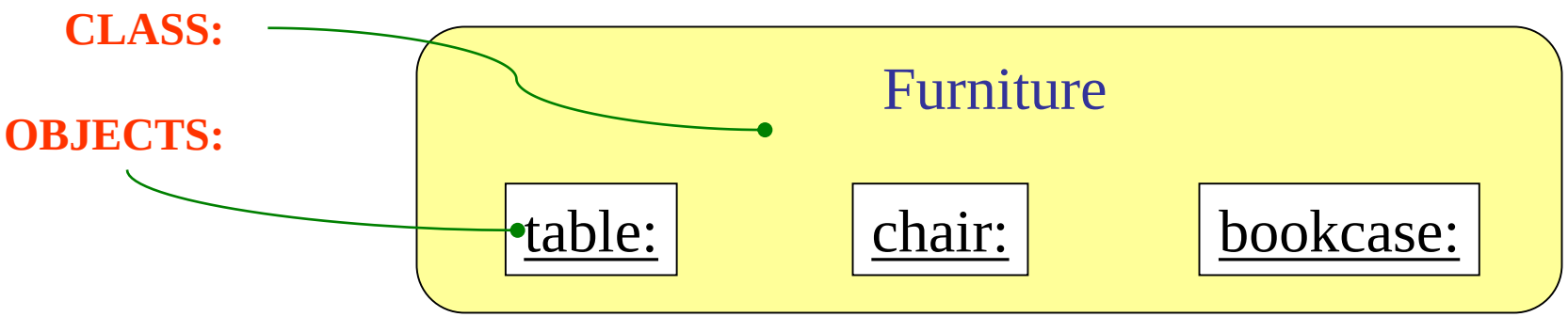
Objects and Classes

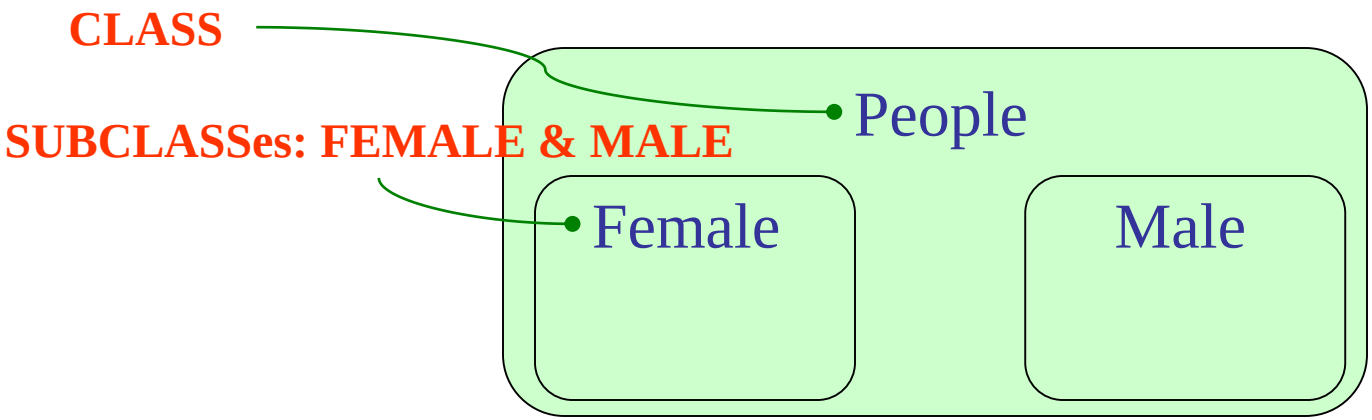
- By watching the objects we recognize that **some objects share the same type of qualities** (a person has a name, the same number of chromosomes, ...)
- **so we might formulate a general concept**
- **– … that is define a Class**.

- This process is called ……

- Abstraction

When you classify objects you decide their type





Class and Object Definitions

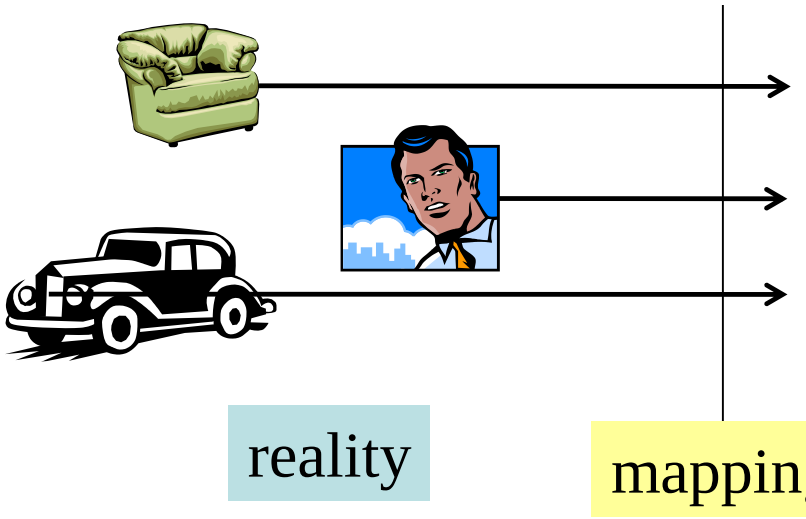
- **Class**:
- A description of a set of objects that share the same attributes, operations, relationships, and semantics.
- **Object**:
- A concrete manifestation of an abstraction
- An entity with a well defined boundary and identity that encapsulates state and behavior
- An instance of a class

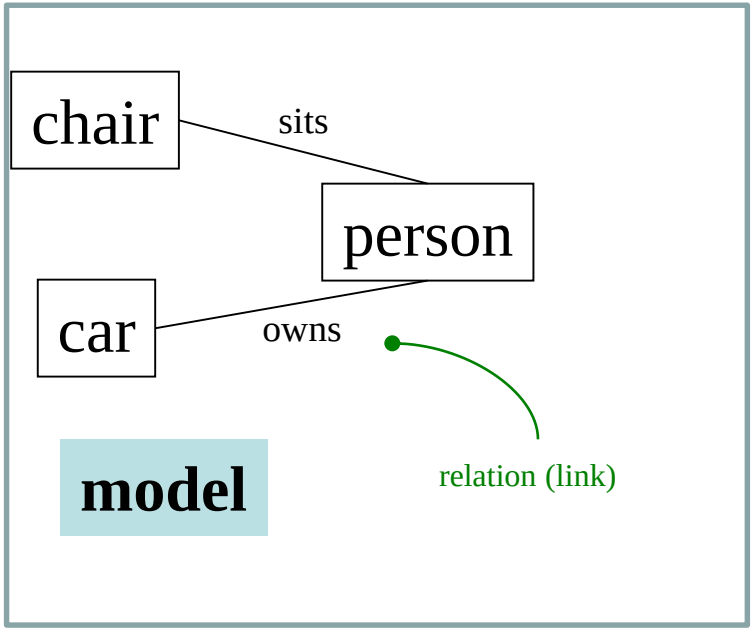
Making a Model

- You have a problem domain
- You want to make a system that **solves some problem in** this **domain**.
- You **single out what is essential for your problem**
- You also typically simplify reality.

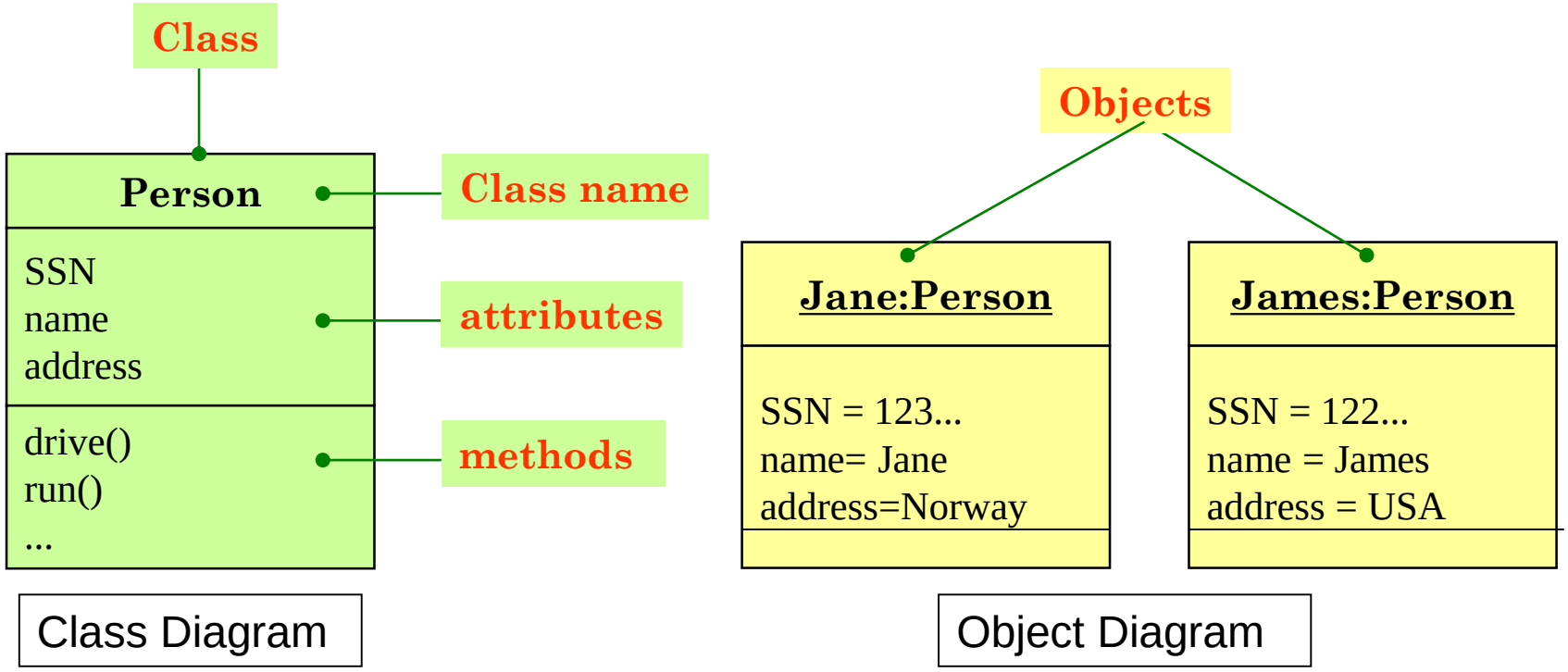
•You make a model of your system, **objects from reality are mapped directly into objects in the model**.

A person sits on a chair. A person owns a car.





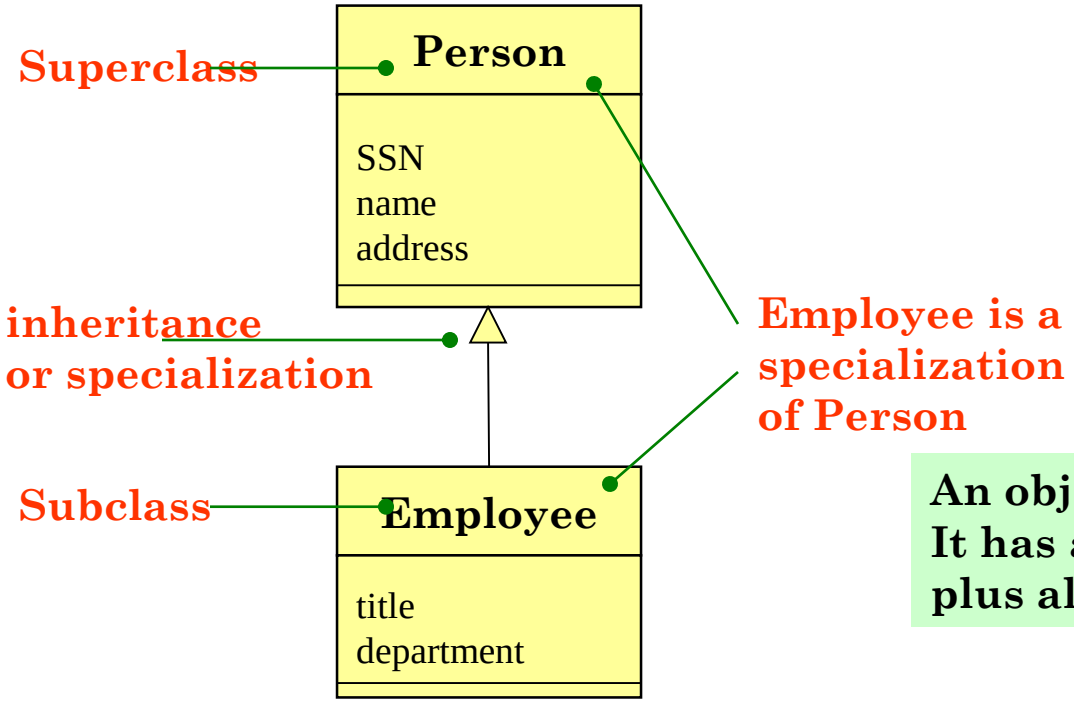
Classes and Objects in UML



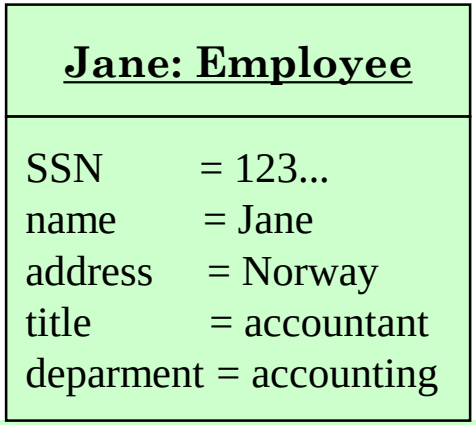
- When you declare the class, you select which attributes and operations are of interest for your system.
- Objects of the given class will have individual values for the specified attributes.

Class Hierarchy ~ Inheritance

- An employee is a special type of person, so if you already have a Person class and need a Employee class then it should be possible to take advantage of the already existing Person class.



Class Diagrams



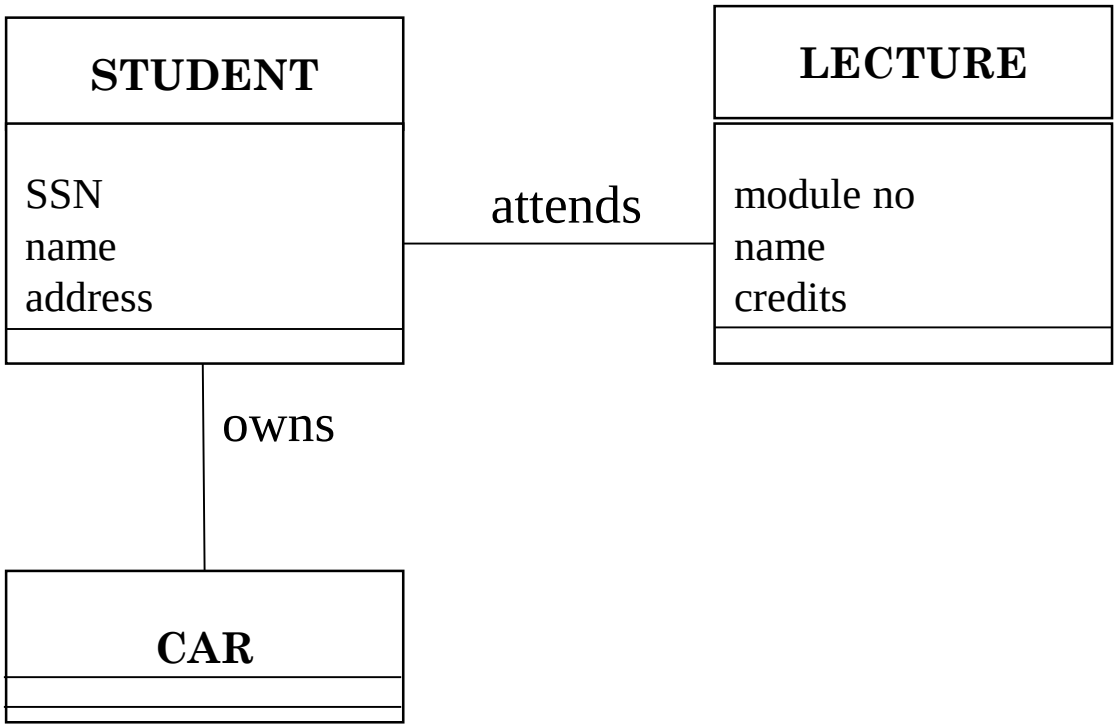
An object of type Employee is also a Person It has all the attributes that a Person has plus all of its own.

Object Diagram

1 - Relationships

Classes are typically connected in some way or another.

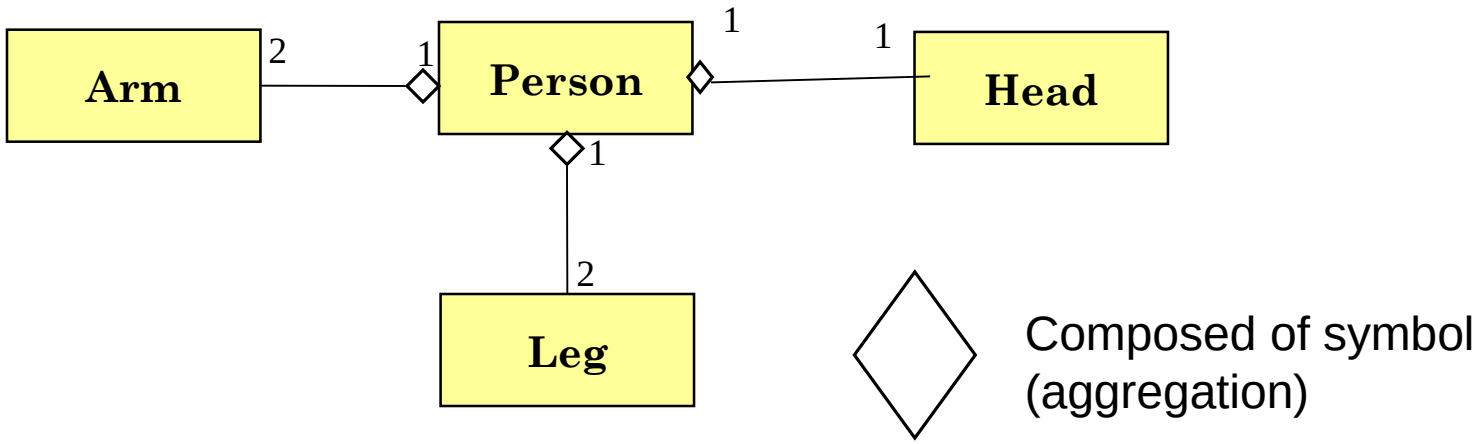
- Example:
- A student attends lectures
- A student owns a car



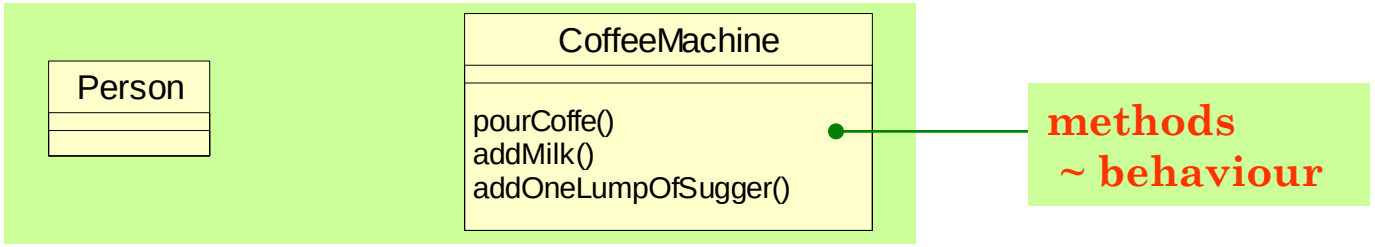
2 - Relationships

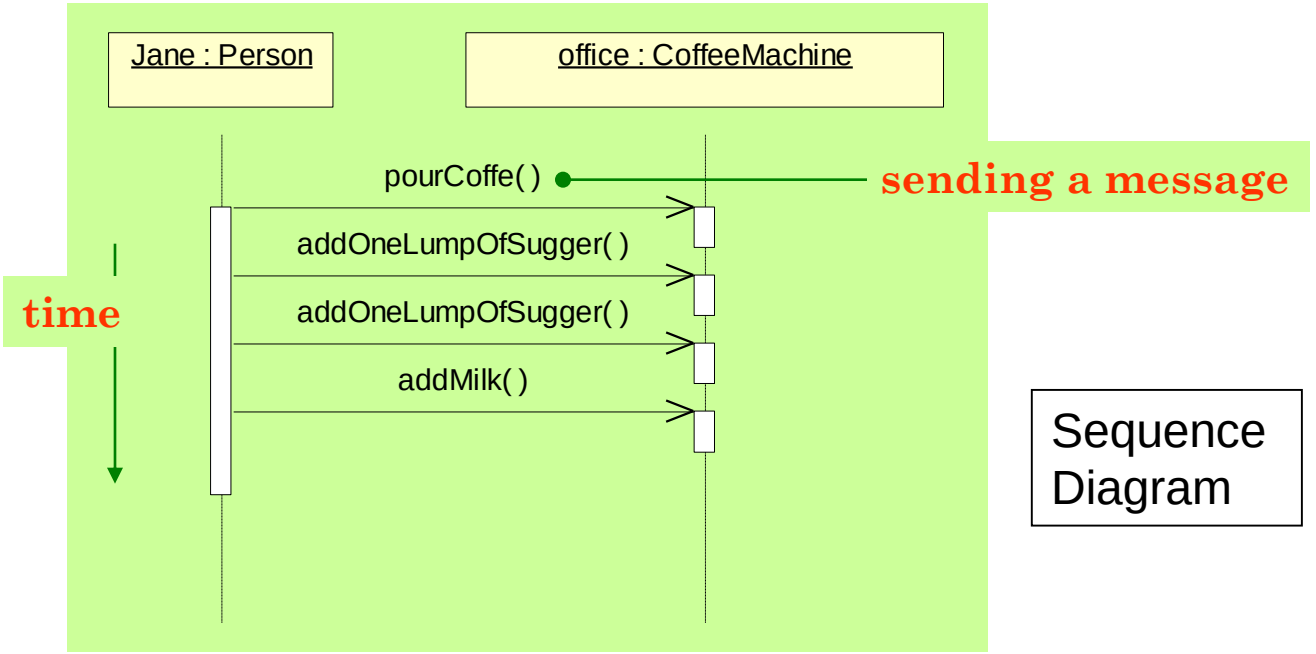
- Some objects can be seen as consisting of other objects.

A person consists of …….2 arms …2 legs ….. 1 head



Objects have Behaviour





Sequence Diagram

Encapsulation / Data Hiding

- Objects are encapsulated so that they share only the elements that are relevant to other objects and hide their own data stores and methods when they are not relevant to other elements
- Encapsulation is achieved by **never letting objects directly access fields** - fields are made private (or protected)
- Fields should only be accessed through the object's methods
- If you want objects of different types to access the fields, then make **public methods** for this purpose.

UML

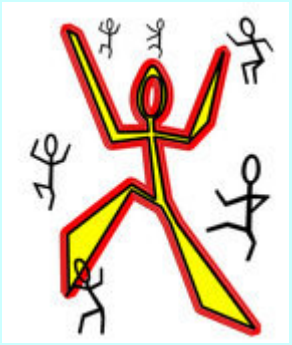
Introduction
&
Overview

Objectives

- To understand why we use model
- To understand why software fails
- To provide some examples of UML diagrams
- To understand the Rational Unified Process (RUP)

A Picture is worth …..

- A picture is worth a 0,000 lines of code
- Stick people oldest form of communication
- Central character in one of newest languages developed



U ~ Unified

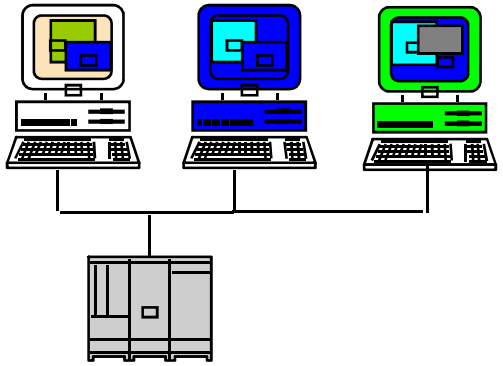
M ~ Modelling

L ~ Language

Why do we need to model?

- *"Modeling captures essential parts of the system"*

James Rumbaugh



What does this represent?

Why are models valuable?

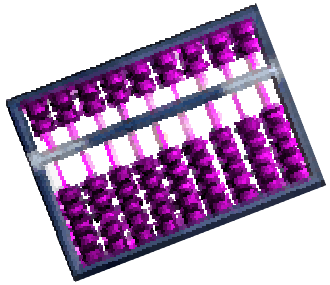
…in solving problems

- Diagrams & pictures
- Cheap
- Fast
- Easy
- Flexible

Understanding UML

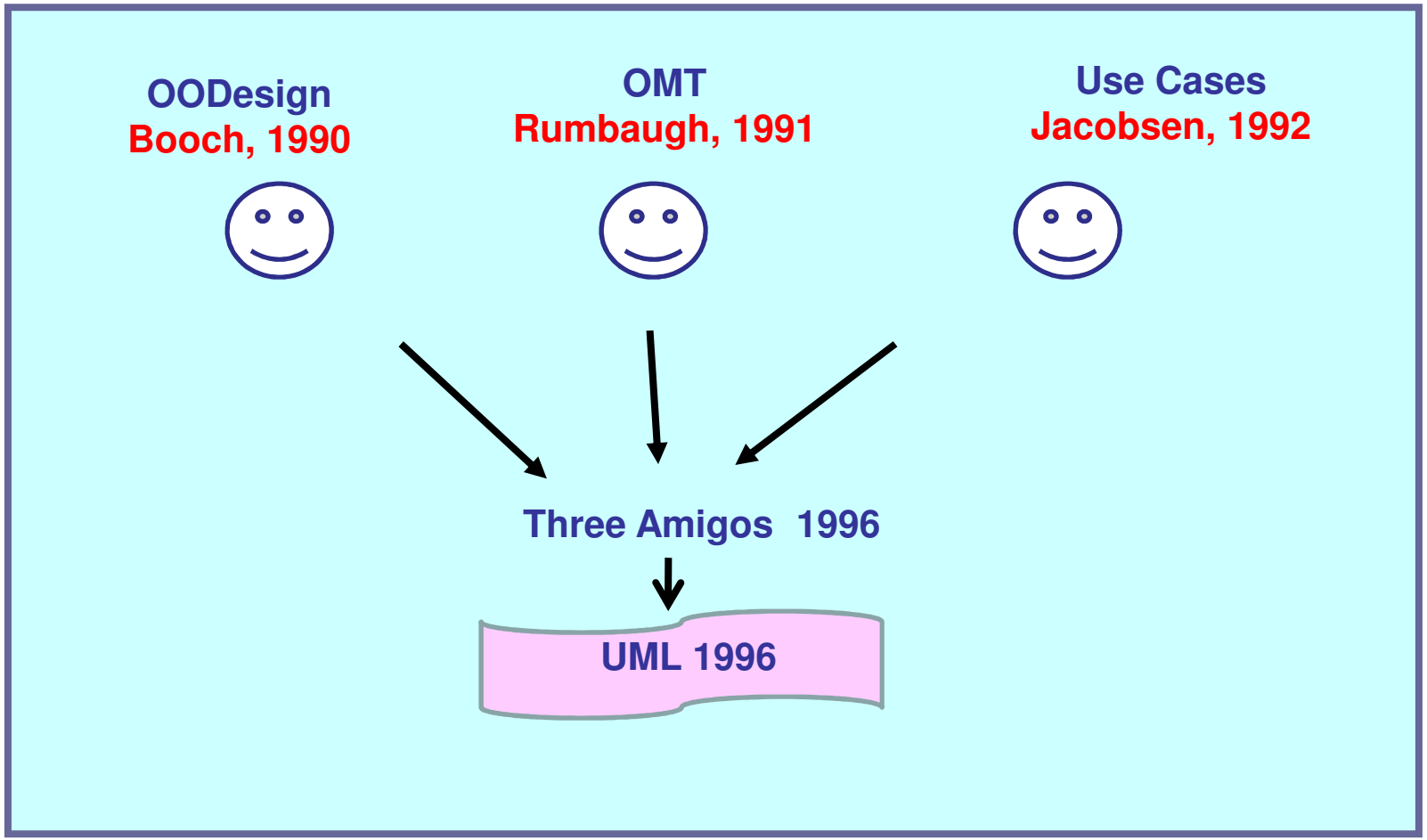
- An official definition of a pictorial language where there are common symbols & relationships
- Language like English or Afrikaans
- Learn the symbols and grammar
- It is a standard
- UML specification defined by consortium of companies that make up OMG
- Lots of tool support

Evolution of Software Design



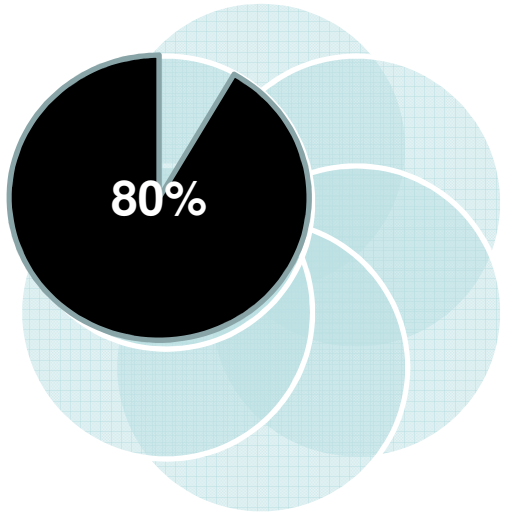
- 5000 yrs ago --- Chinese ---- 1st computer --- Abacus
- 150 yrs ago ---Charles Babbage --- computing machine
- 1940 --- Alan Turing --- Turing computing machine
- 1960 ----Punch cards
- 1970 ----Cobol --- Spaghetti Code – Structured Analysis & Design
- 1960s – OO language – Smalltalk
- 1986 – C++
- 1980s – PC
- 1990s – UML developed - Three Amigos

Evolution of UML



Why are models valuable?

80% of all software **projects** fail



Late

Never delivered

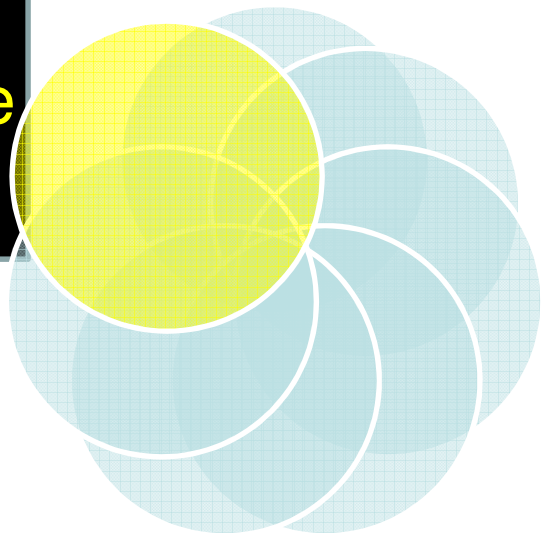
Exceed their budgets

Don't provide features customers **need** or desire

Why are models valuable?

80% of all software **projects** fail

More **time** needs to be spent on software analysis and design



Late

Exceed their budgets

Never delivered

Don't provide features customers **need** or desire

Modeling and Future of S/w Development

Emphasis on software analysis and design

- Fewer projects will fail
- Software quality will improve

More software engineers will be expected to learn UML

Using Models

- Models consist of diagrams & pictures
- Cheaper to produce and experiment with than code
- More information conveyed in a picture
- Easier to change picture than code
  - no attachment
- Models use simple symbols
  - Easier to understand
  - More stakeholders participate in design of system

Creating Diagrams

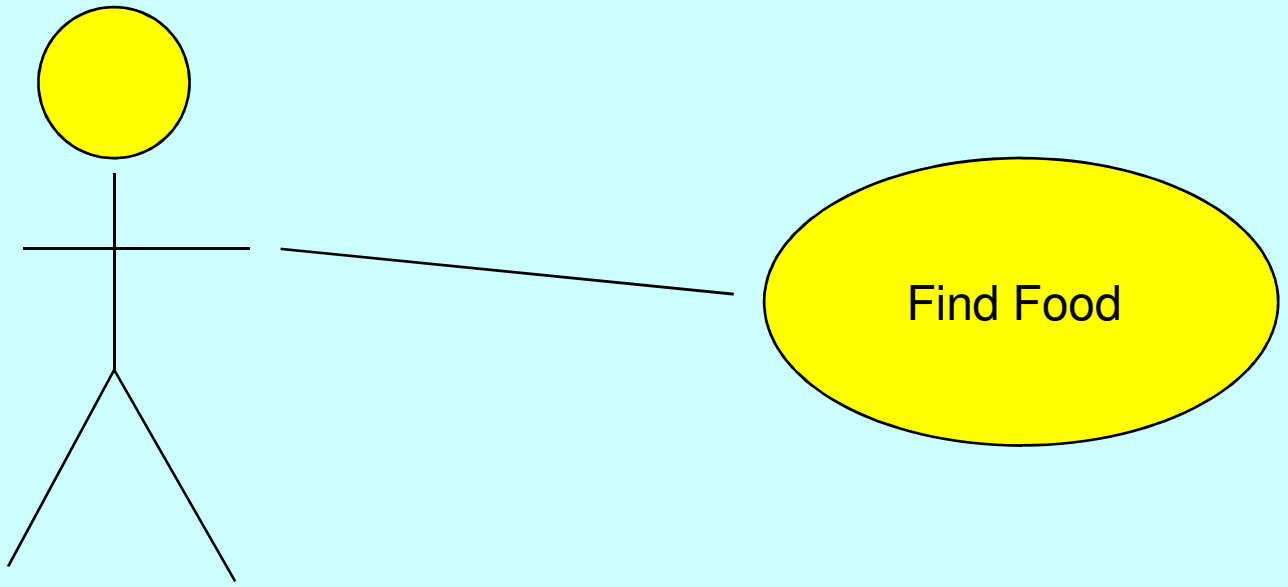
- Capture the important parts of the problem and the solution accurately
- Can create several types of diagrams
  - Convey different kinds of information
- Goal is to be accurate
- Add text to diagrams to clarify
  - not too much
- Get a second opinion

Examples of UML Diagrams

- Use Case
- Class
- Activity
- Interaction
- Sequence
- Collaboration
- State

Use Case Diagram

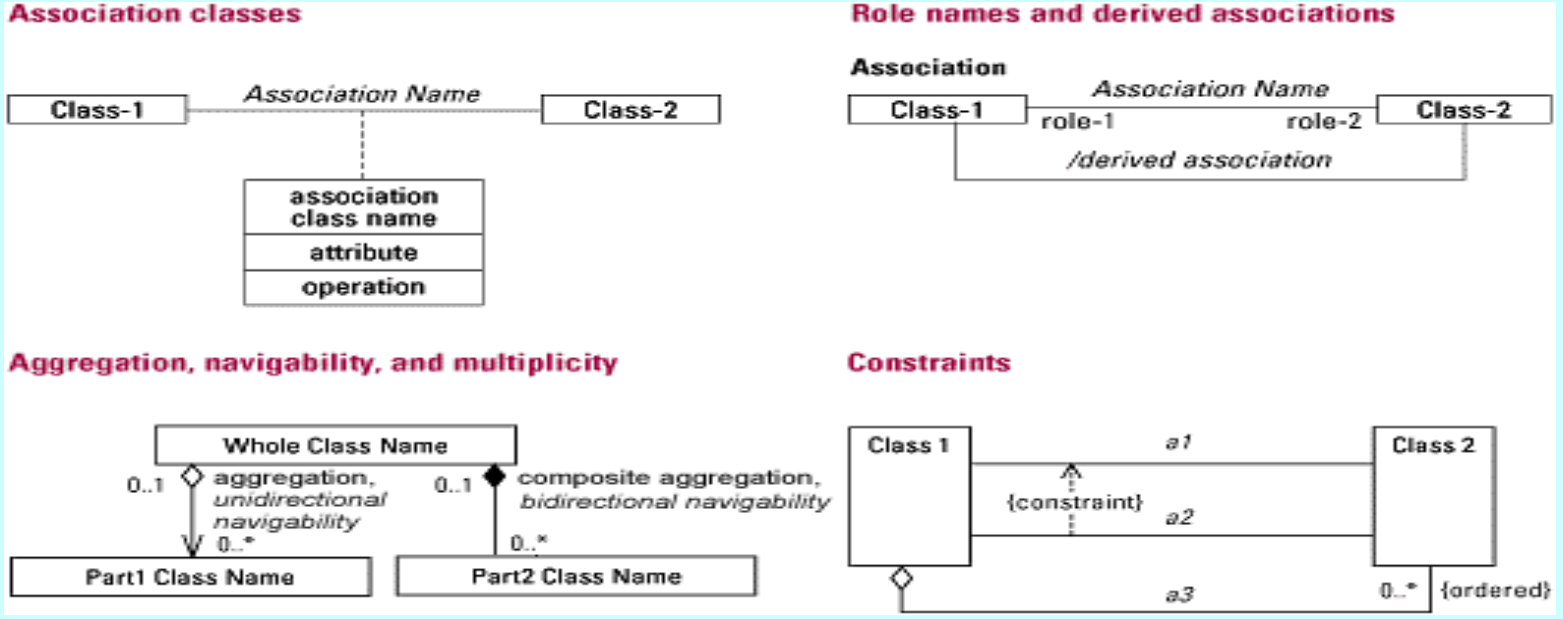
- Show the macro requirements of the system



‘Find Food’ use case

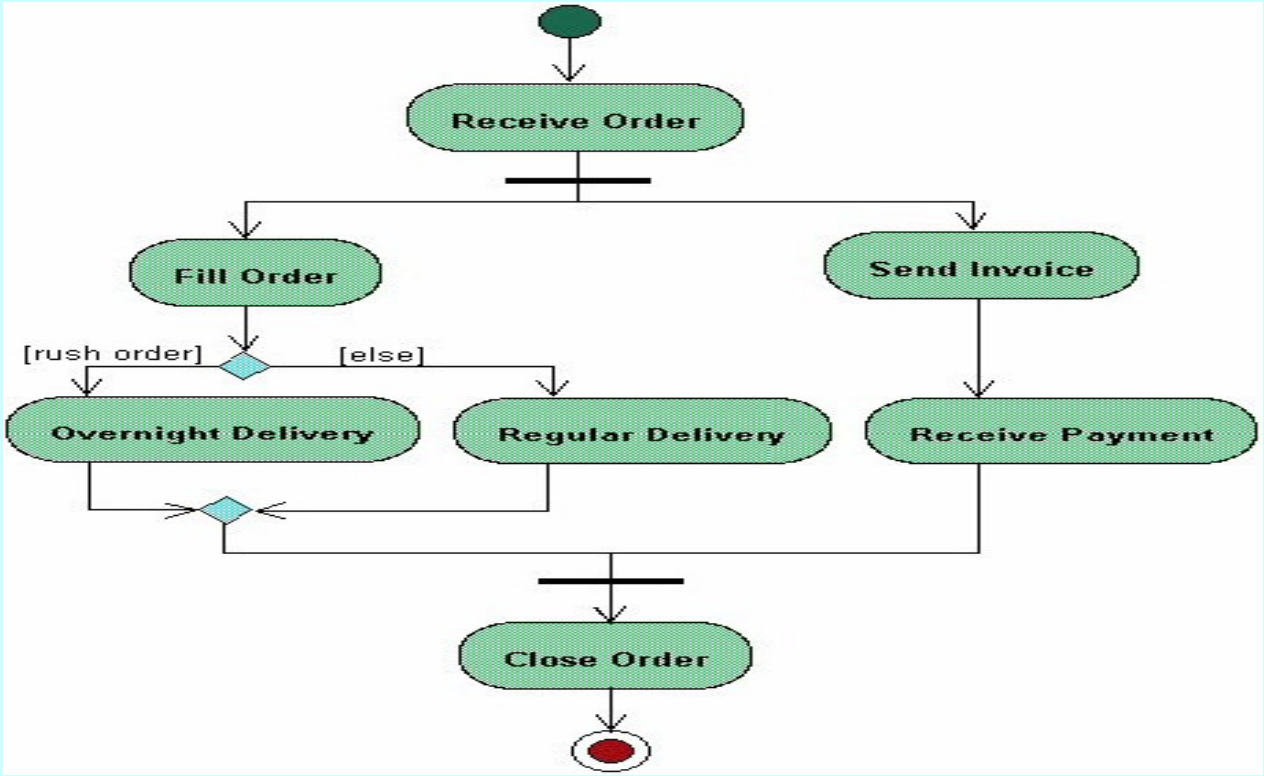
Class Diagram

- Show the classes in the system and the relationships between those classes
- Show a static view of the system



Activity Diagram

- UML version of a flowchart
- Help to understand the problem

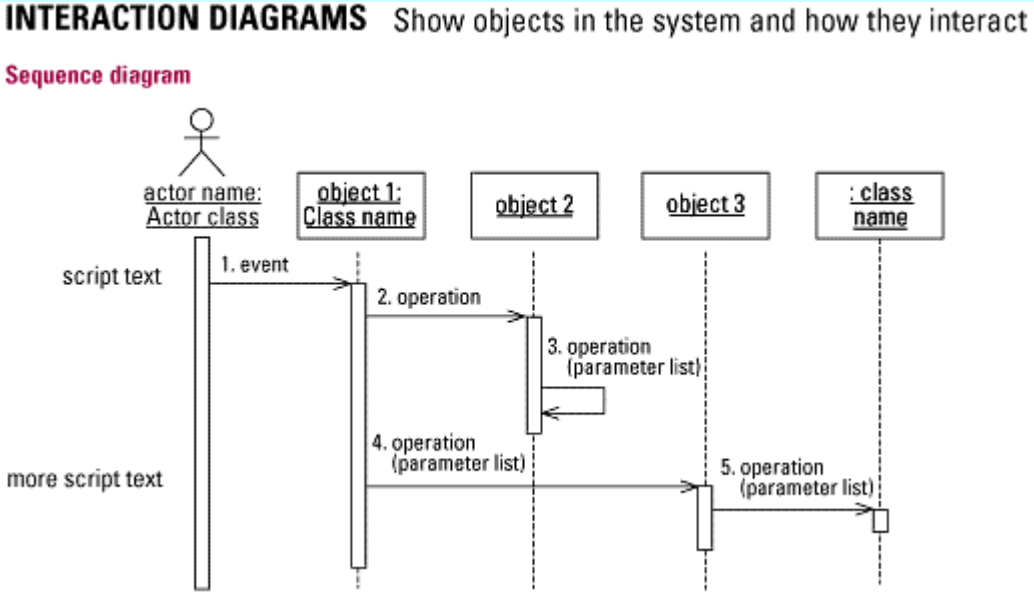


Interaction Diagram

- Two types
  1. Sequence Diagram and
  2. Collaboration Diagram
- Convey the same information from different perspectives
- Generally a sequence diagram is easier to read.

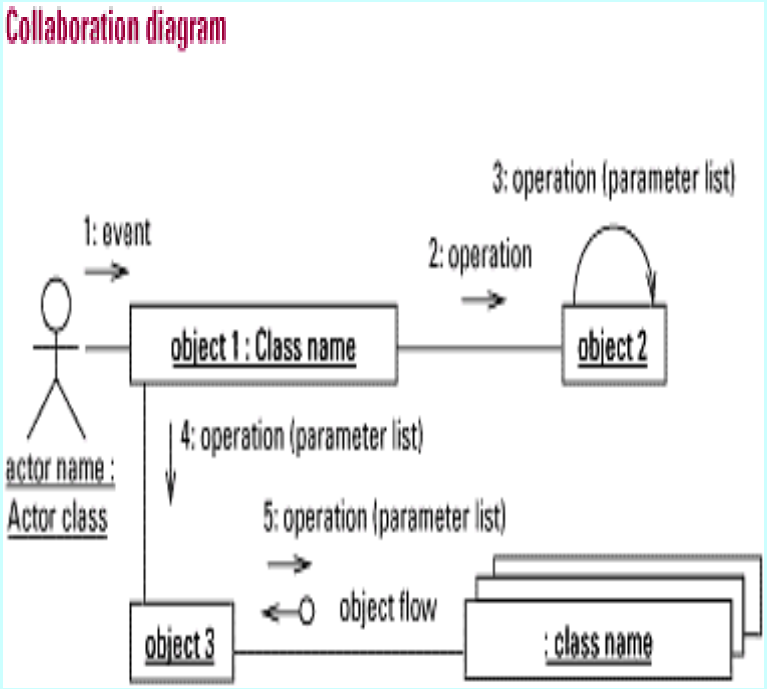
Sequence Diagram

- Show classes along the top and messages sent between the classes
- Can follow the messages sent over time from top left to bottom right



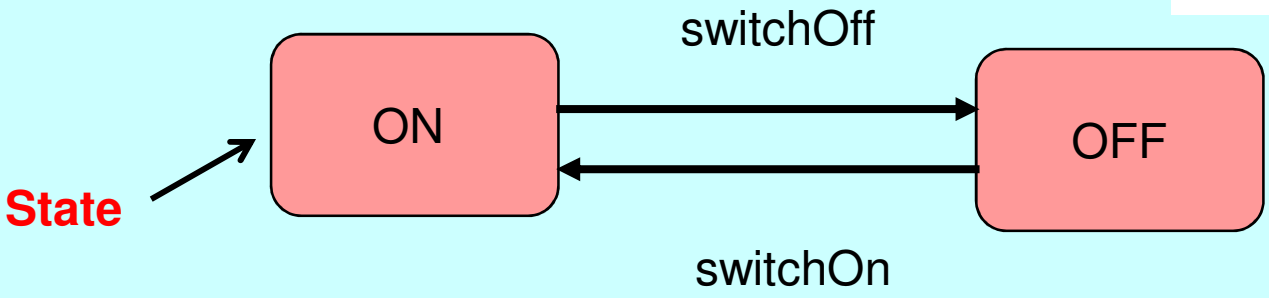
Collaboration Diagram

- Use the same classes and objects as sequence diagram
- Number the messages to indicate the order they occur



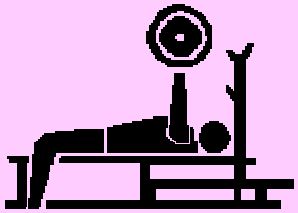
State Diagram

- Shows the changing state of a single object as it passes through a system



Exercise

- Why do many software project fail?

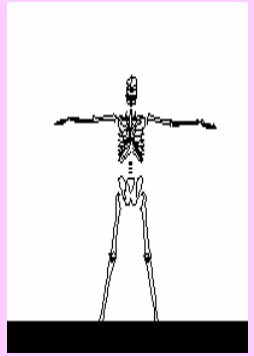


- Where should a software developer put a lot of their effort into?

---------- & ----------

Exercise

- What are the 3 guys who developed UML called?



- What is an Activity Diagram like?

- What states can a car be in?

--- ---

--- ---