WIA2002: Software Modelling
Semester 1, Session 2016/17

Lecture 3: Modelling Concepts

Learning objectives

- Know what is a model.
- Understand the distinction between a model and a diagram.
- Know the UML concept of a model.
- Know the fundamental concepts of object-orientation, including:
    - Objects and classes.
    - Generalization, specialization and inheritance.
    - Information hiding and message passing.
- Understand why object-oriented approach is used.
- Know the SADT diagrams.
- Know the difference between object-oriented and structured approach.

What is a Model?

- An abstract representation of something real or imaginary.
- Like a map, a model represents something else.
- A useful model has the right level of detail and represents only what is important for the task in hand.
- Many things can be modelled: bridges, traffic flow, buildings, economic policy.

Why use a Model?

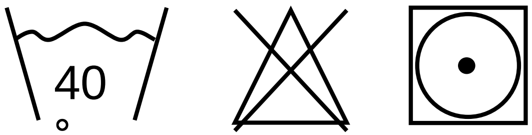
- A model is quicker and easier to build than the real thing.
- A model can be used in a simulation.
- A model can evolve as we learn.
- We can choose which details to include in a model.
- A model can represent real or imaginary things from any domain.

Modelling Organizations

- Organizations are human activity systems.
- The situation is complex.
- Stakeholders have different views.
- We have to model requirements accurately, completely and unambiguously.
- The model must not prejudge the solution.

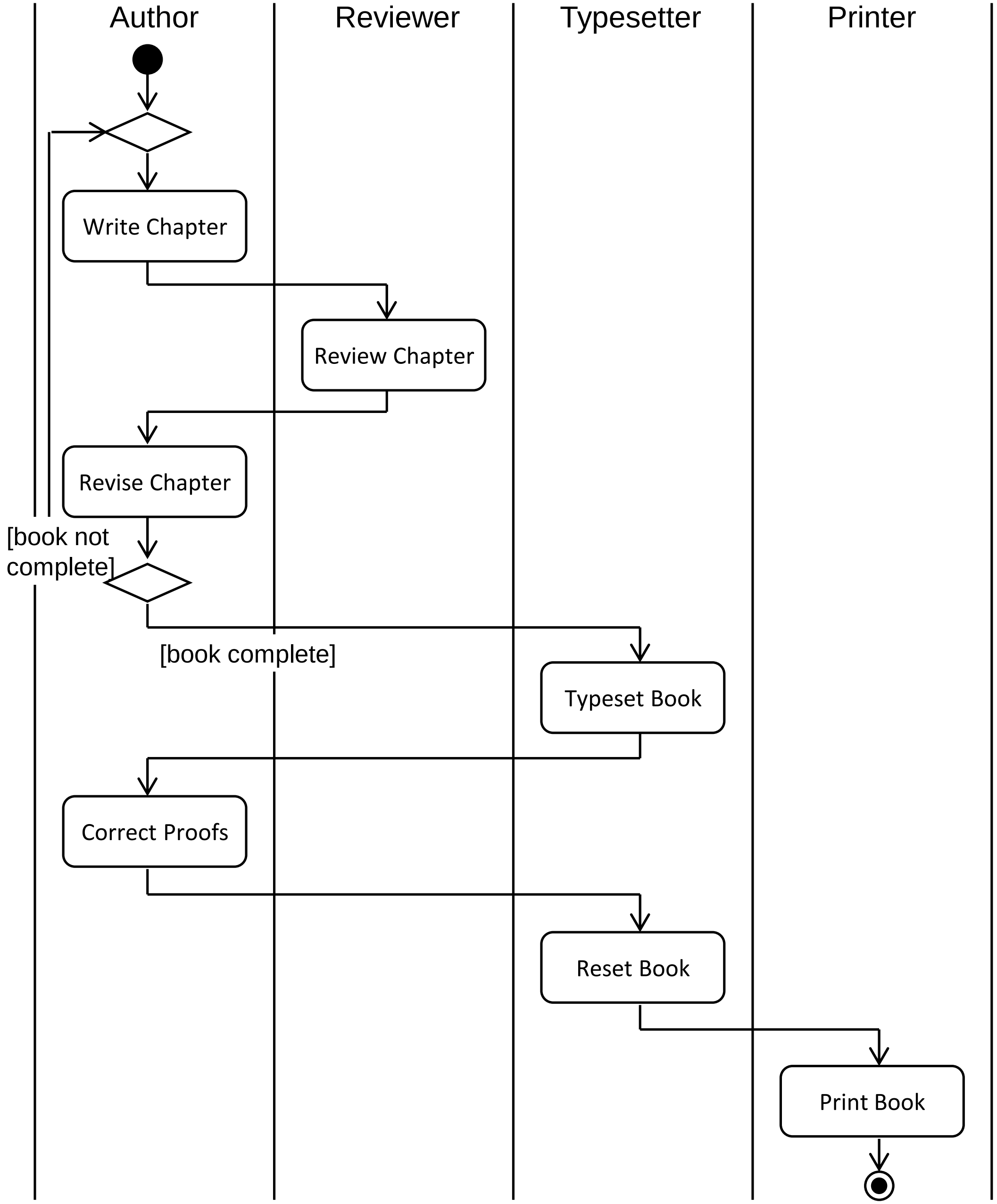
What is a Diagram?

- Abstract shapes are used to represent things or actions from the real world.
- Diagrams follow rules or standards.
- The standards make sure that different people will interpret the diagram in the same way.

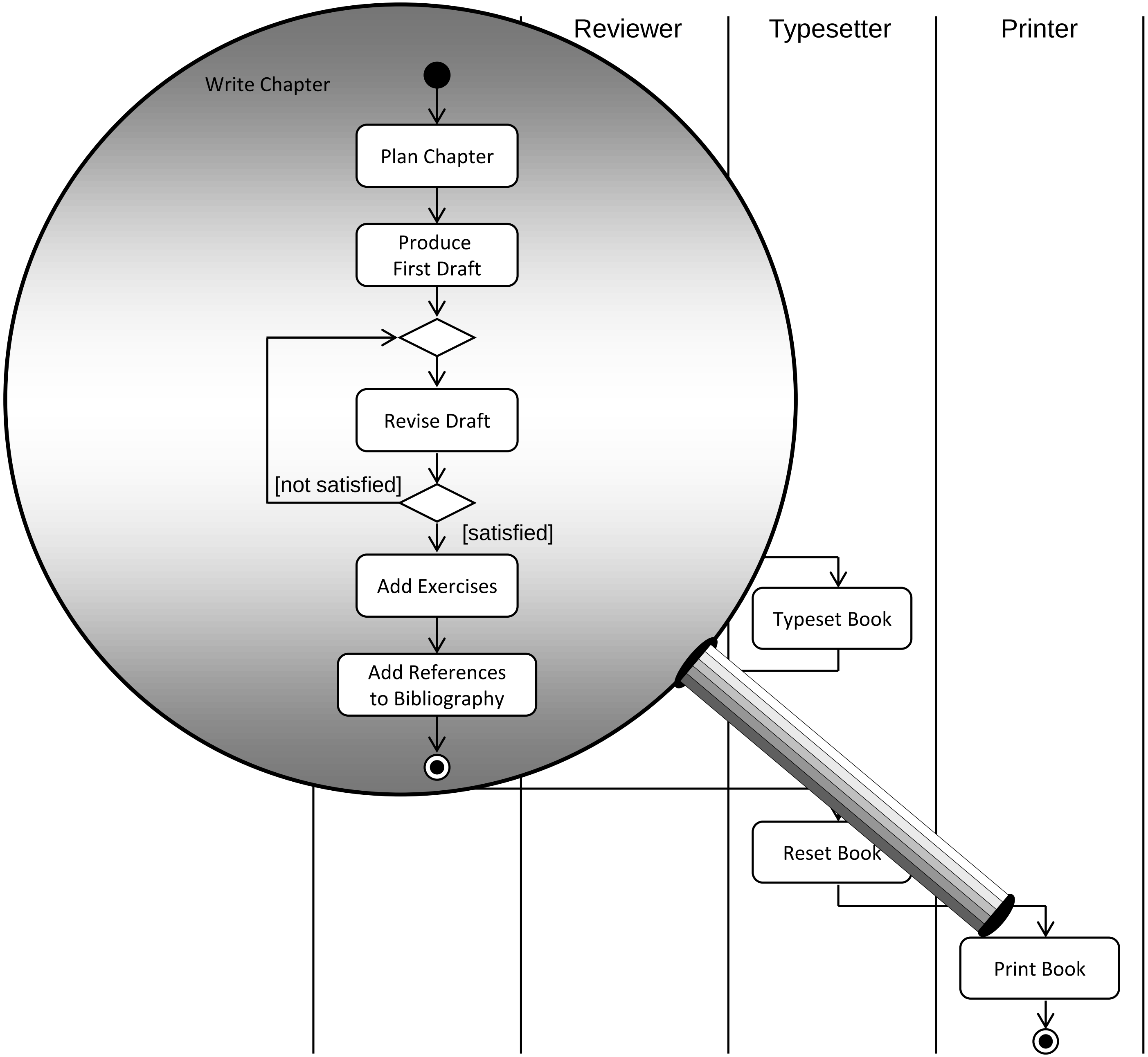


An Example of a Diagram

- An activity diagram of the tasks involved in producing a book.

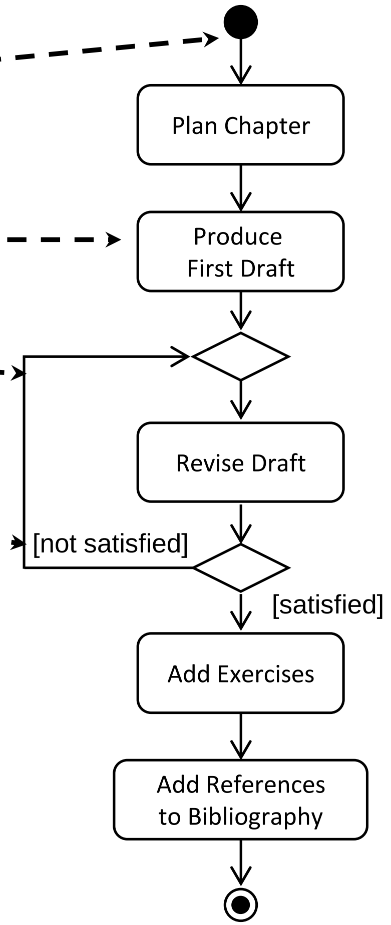


Hiding Detail



Diagrams in UML

- UML diagrams consist of:
  - icons
  - two-dimensional symbols
  - paths
  - Strings



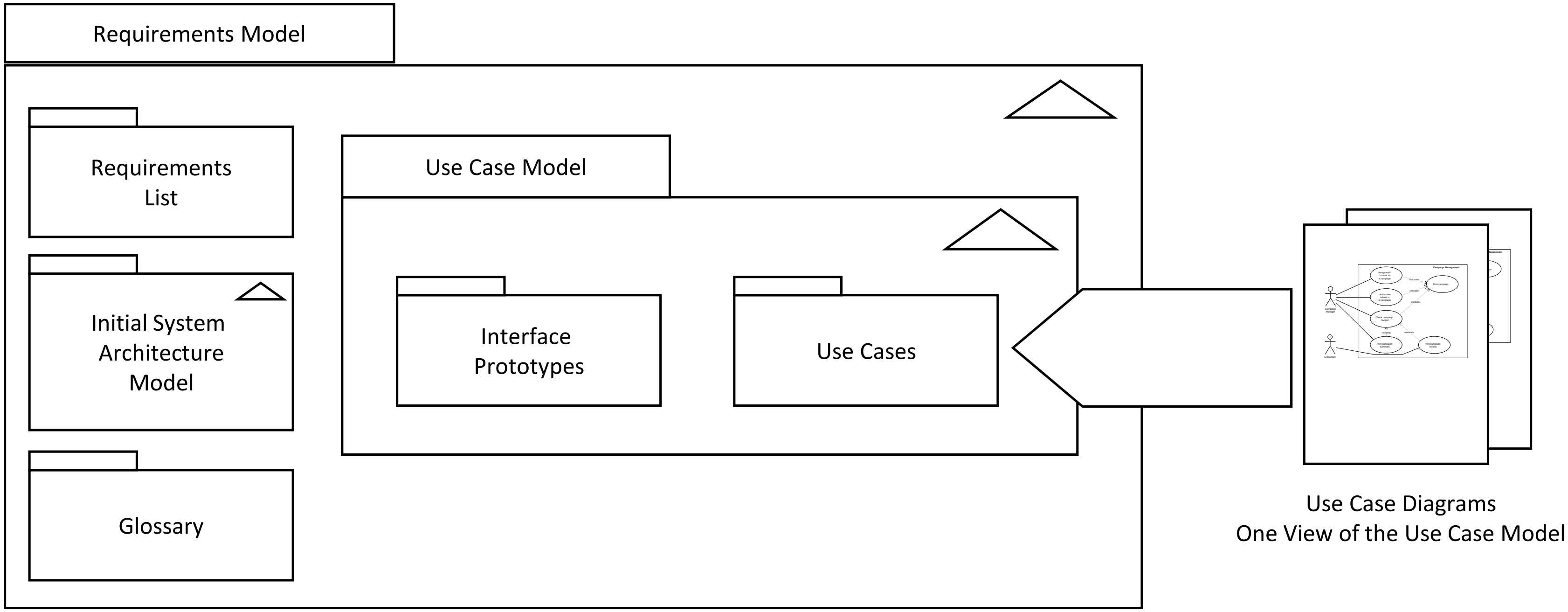
- UML diagrams are defined in the UML specification.

Diagrams vs. Models

- A diagram illustrates some aspect of a system.
- A model provides a complete view of a system at a particular stage and from a particular perspective.
- A model may consist of a single diagram, but most consist of many related diagrams and supporting data and documentation.

Relationship between Models and Diagrams

- Use Case Diagrams are one view of the Use Case Model in the Requirements Model.



Examples of Models

- Requirements Model
  - complete view of requirements.
  - may include other models, such as a Use Case Model.
  - includes textual description as well as sets of diagrams.

Examples of Models

- Behavioural Model
  - shows how the system responds to events in the outside world and the passage of time.
  - an initial model may just use Communication Diagrams.
  - a later model will include Sequence Diagrams and State Machines.

Models in UML

- 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.
- A model is complete and consistent at the chosen level of abstraction.

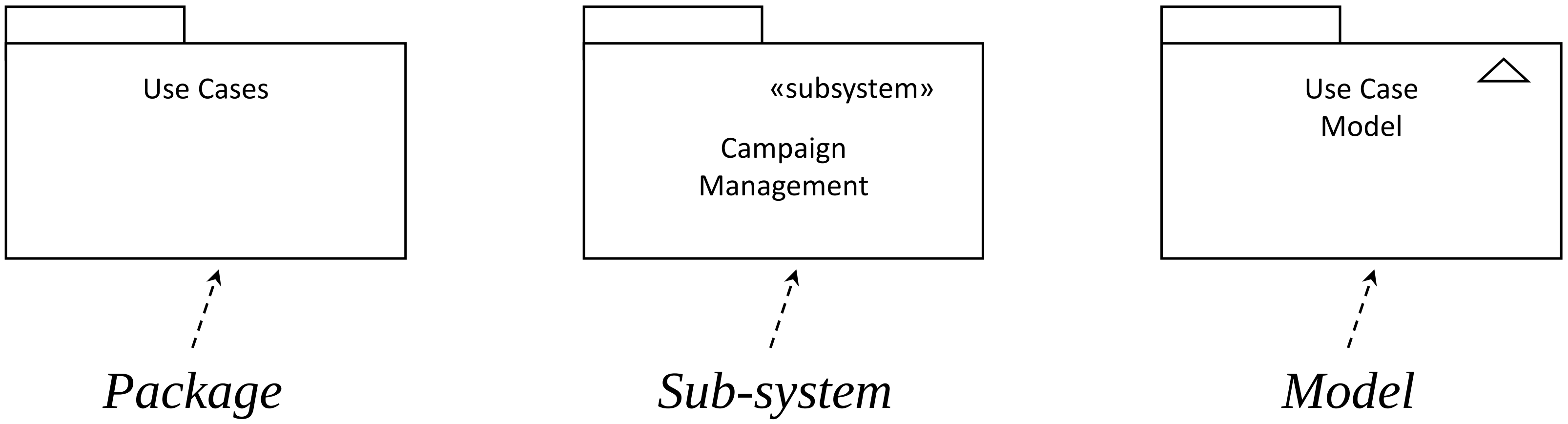
Models in UML

- Different models present different views of the system, for example:
  - use case view
  - design view
  - process view
  - implementation view
  - deployment view

(Booch et al., 1999)

Packages, Sub-systems and Models

- UML has notation for showing subsystems and models, and also for packages, which are a mechanism for organising models (e.g. in CASE tools).

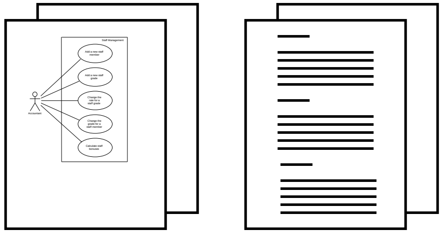


Developing Models

- 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.

Development of the Use Case Model

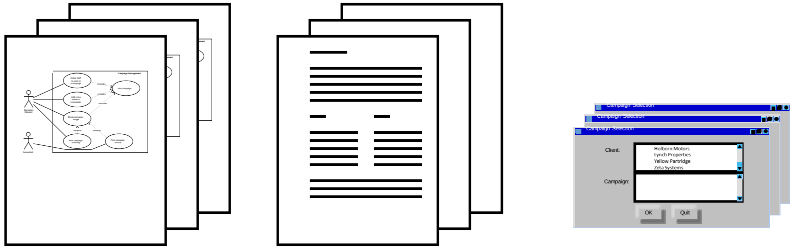
Iteration 1
Obvious use cases.
Simple use case descriptions.



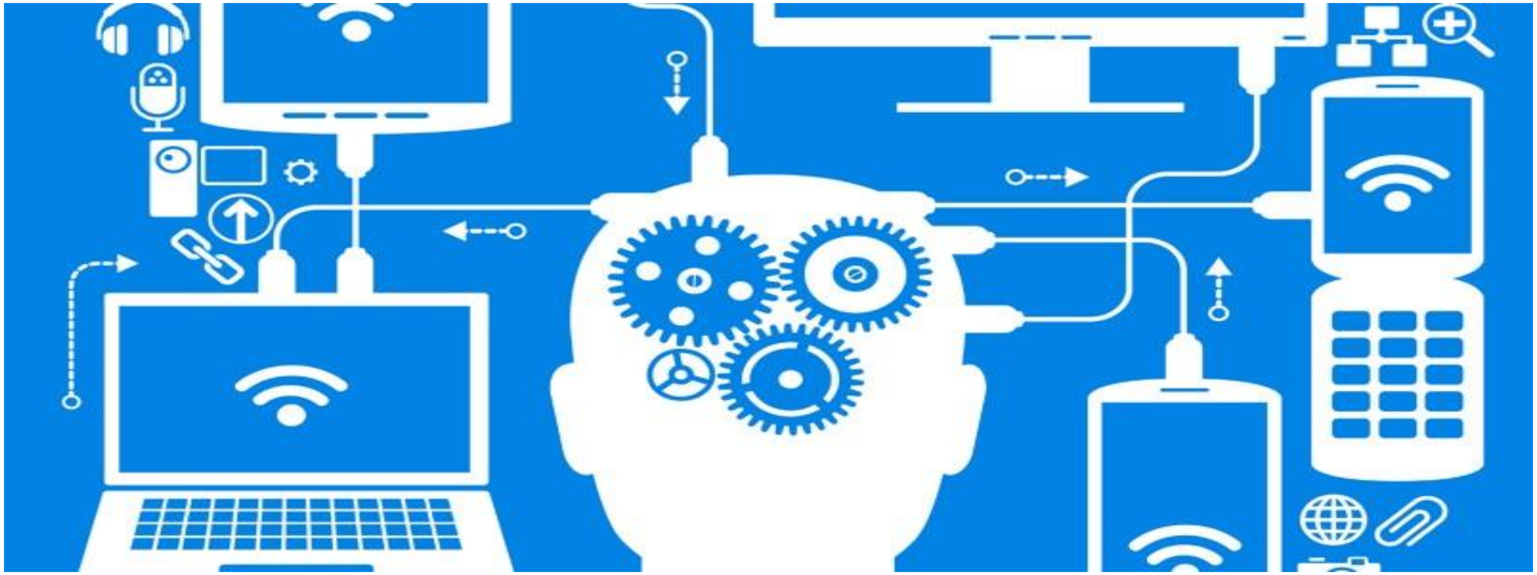
Iteration 2
Additional use cases.
Simple use case descriptions.
Prototypes.

Campaign Selection
Client:
Holborn Motors
Lynch Properties
Yellow Partridge
Zeta Systems
Campaign:
OK
Quit

Iteration 3
Structured use cases.
Structured use case descriptions.
Prototypes.



Any question so far..?



Objects

An object is:

> “an abstraction of something in a problem domain, reflecting the capabilities of the system to
>
> - keep information about it,
> - interact with it,
> - or both.”

Coad and Yourdon (1990)

Objects

> “Objects have state, behaviour and identity.”

Booch (1994)

- *State*: the condition of an object at any moment, affecting how it can behave.

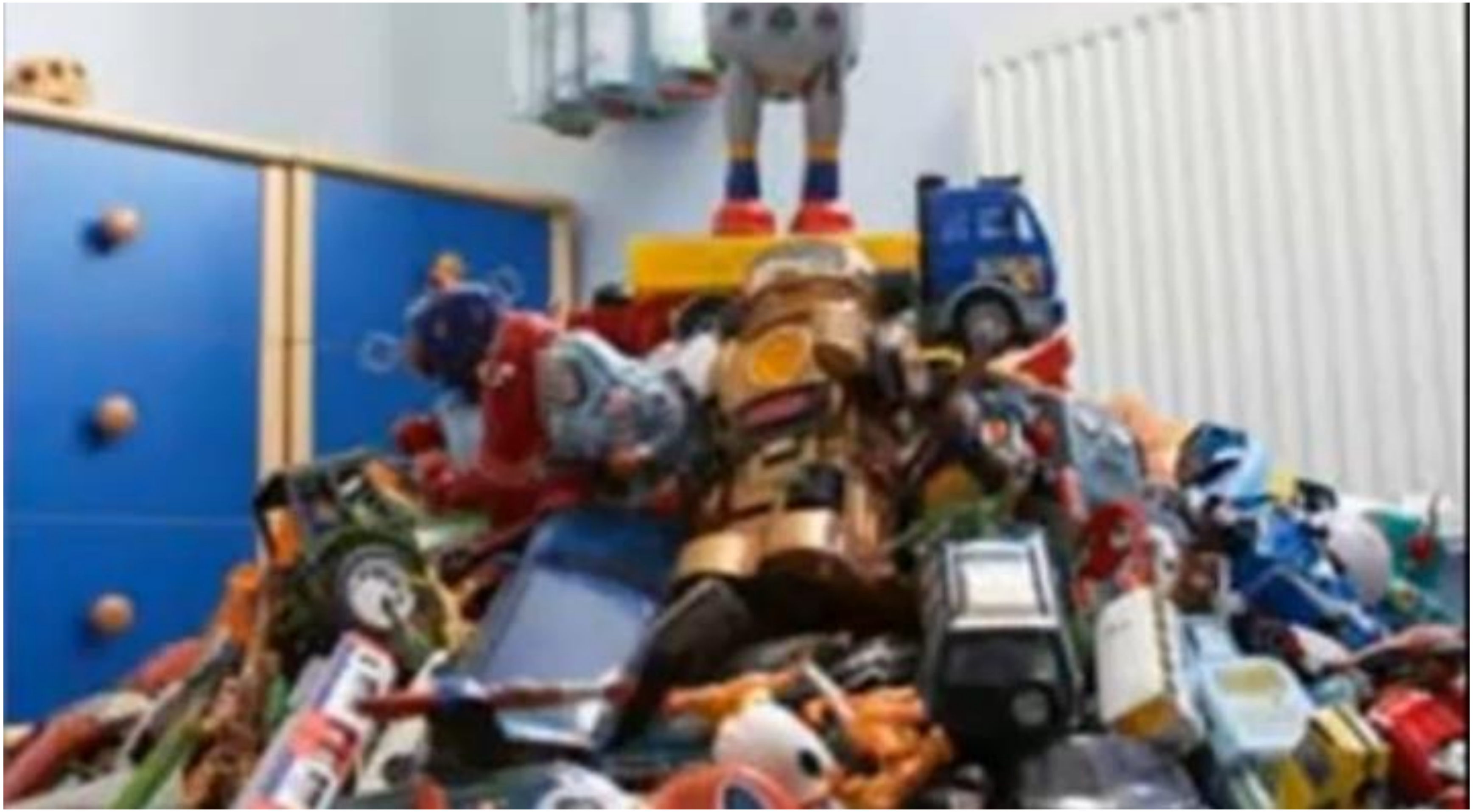
- *Behaviour*: what an object can do, how it can respond to events and stimuli.

- *Identity*: each object is unique.

Examples of Objects

Object	Identity	Behaviour	State
A person.	‘Hussain Pervez.’	Speak, walk, read.	Studying, resting, qualified.
A shirt.	My favourite button white denim shirt.	Shrink, stain, rip.	Pressed, dirty, worn.
A sale.	Sale no #0015, 18/05/05.	Earn loyalty points.	Invoiced, cancelled.
A bottle of ketchup.	*This* bottle of ketchup.	Spill in transit.	Unsold, opened, empty.

Find a green car



Is it better to find it now?

Class and Instance

- All objects are *instances* of some *class.*

- A Class is a description of a set of objects with similar:
  - features (attributes, operations, links);
  - semantics;
  - constraints (e.g. when and whether an object can be instantiated).

OMG (2009)

Class and Instance

- An object is an instance of some class.

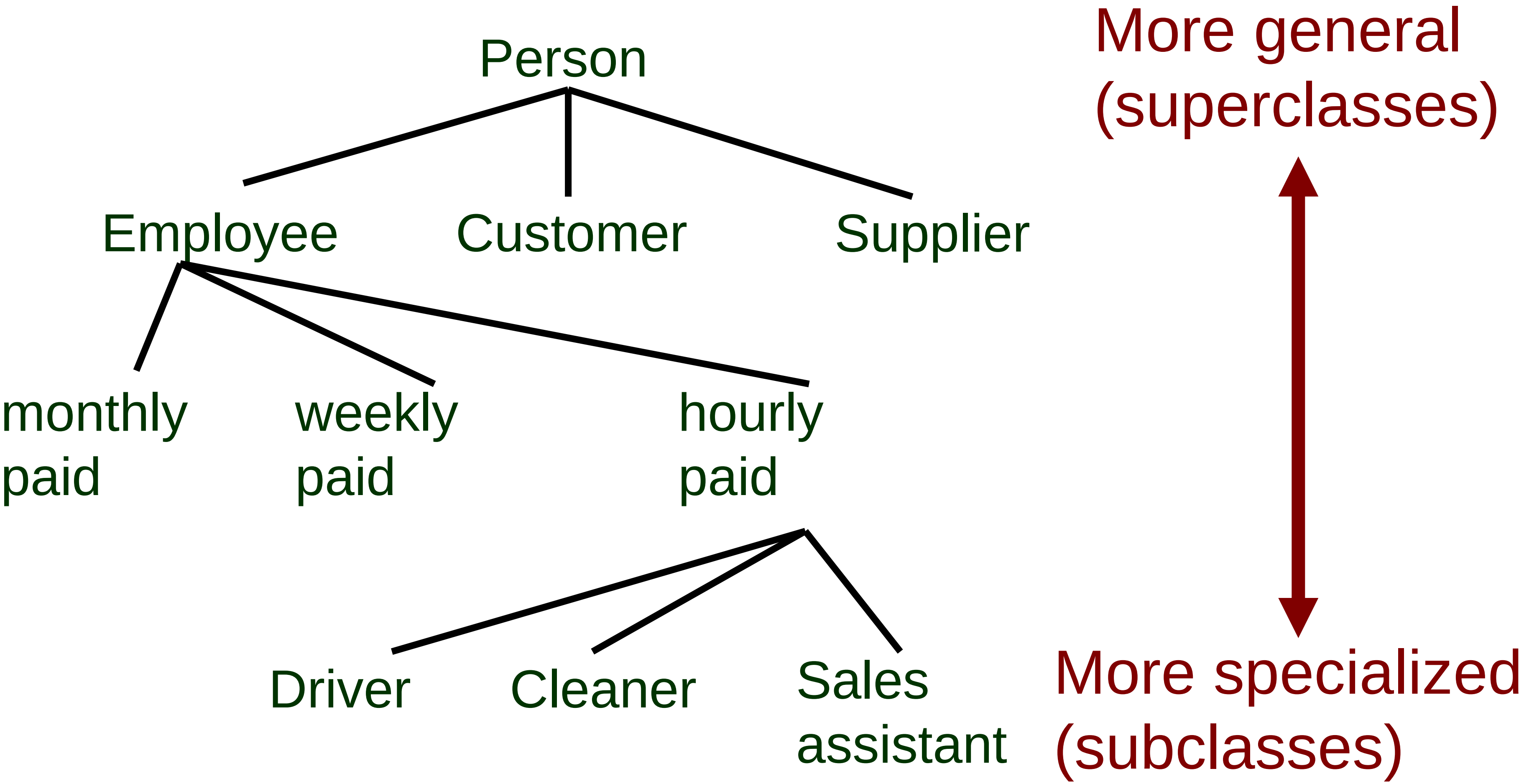
- So, instance = object
  - but also carries connotations of the class to which the object belongs.

- Instances of a class are similar in their:
  - *Structure*: what they *know*, what information they hold, what links they have to other objects.
  - *Behaviour*: what they *can do.*

Generalization and Specialization

- Classification is hierarchic in nature.
- For example, a person may be an employee, a customer, a supplier of a service.
- An employee may be paid monthly, weekly or hourly.
- An hourly paid employee may be a driver, a cleaner, a sales assistant.

Specialization Hierarchy



Person

Employee

Customer

Supplier

monthly paid

weekly paid

hourly paid

Driver

Cleaner

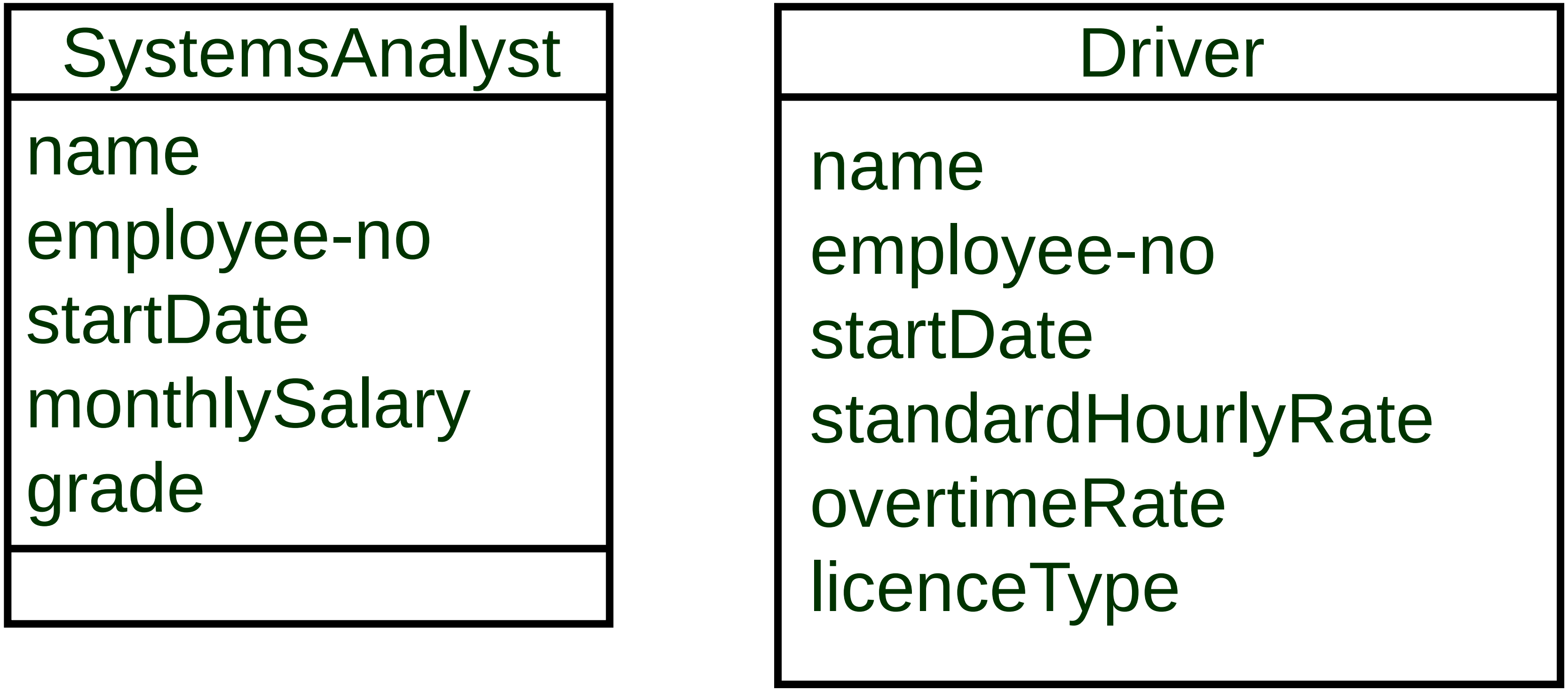
Sales assistant

More general (superclasses)

More specialized (subclasses)

Generalization and Specialization

- More general bits of description are *abstracted out* from specialized classes:

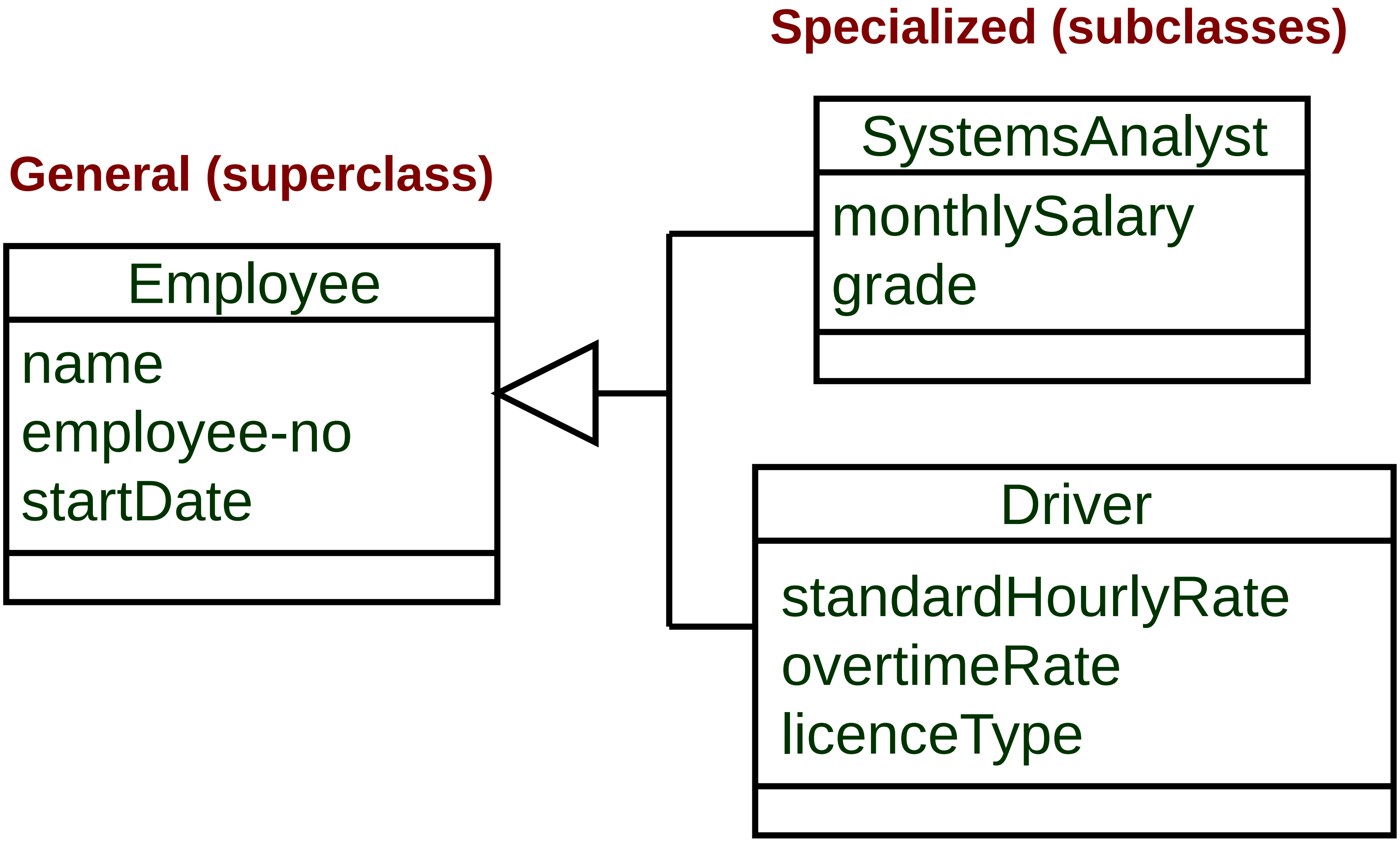


SystemsAnalyst

name
employee-no
startDate
monthlySalary
grade

Driver

name
employee-no
startDate
standardHourlyRate
overtimeRate
licenceType



Specialized (subclasses)

General (superclass)

Employee

name
employee-no
startDate

SystemsAnalyst

monthlySalary
grade

Driver

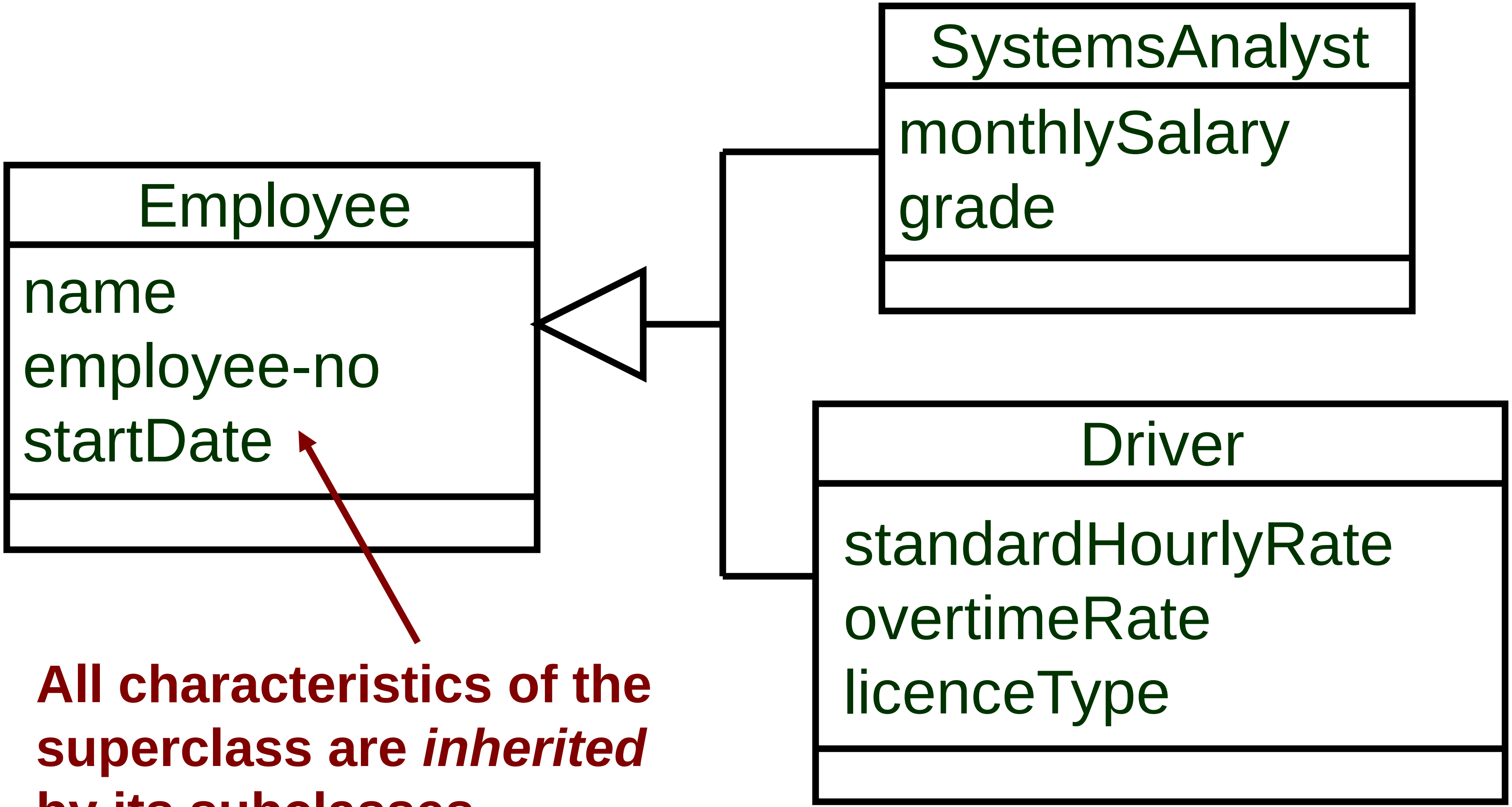
standardHourlyRate
overtimeRate
licenceType

Inheritance

- The *whole* description of a superclass applies to *all* its subclasses, including:
  - Information structure (including associations).
  - Behaviour.

- Often known loosely as *inheritance.*

- (But actually inheritance is how an O-O programming language *implements* generalization / specialization)



Employee

name
employee-no
startDate

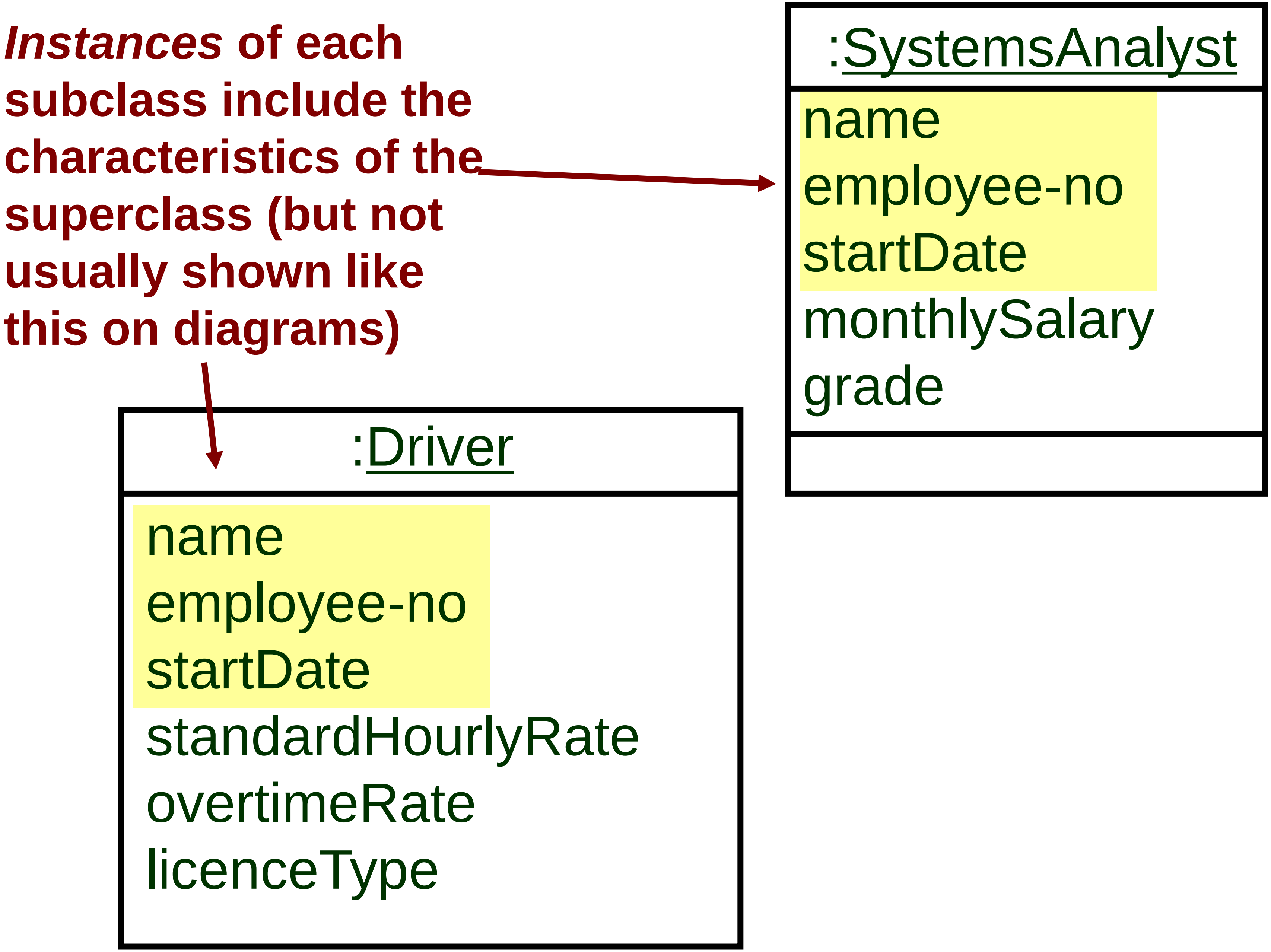
SystemsAnalyst

monthlySalary
grade

Driver

standardHourlyRate
overtimeRate
licenceType

All characteristics of the superclass are *inherited* by its subclasses



Instances **of each subclass include the characteristics of the superclass (but not usually shown like this on diagrams)**

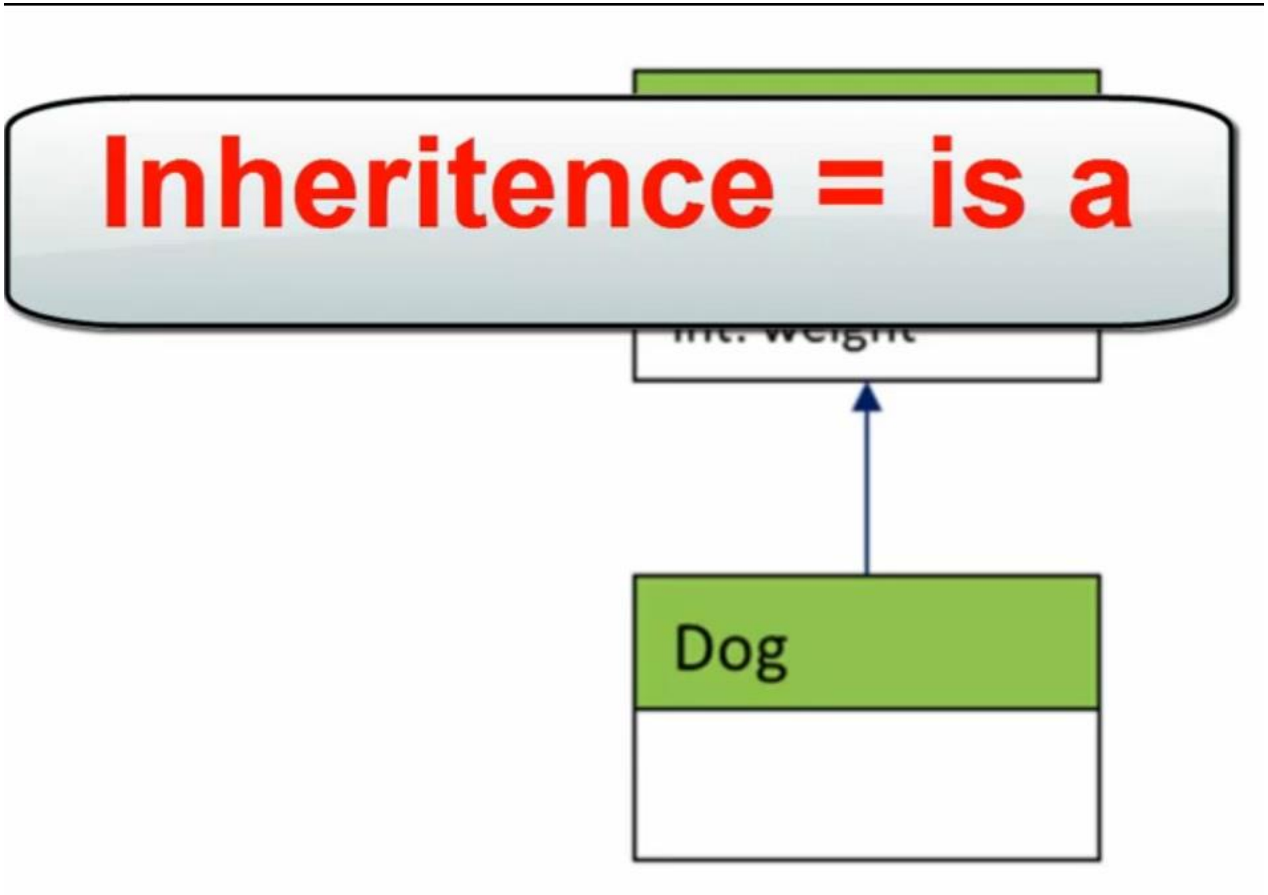
:SystemsAnalyst

name
employee-no
startDate
monthlySalary
grade

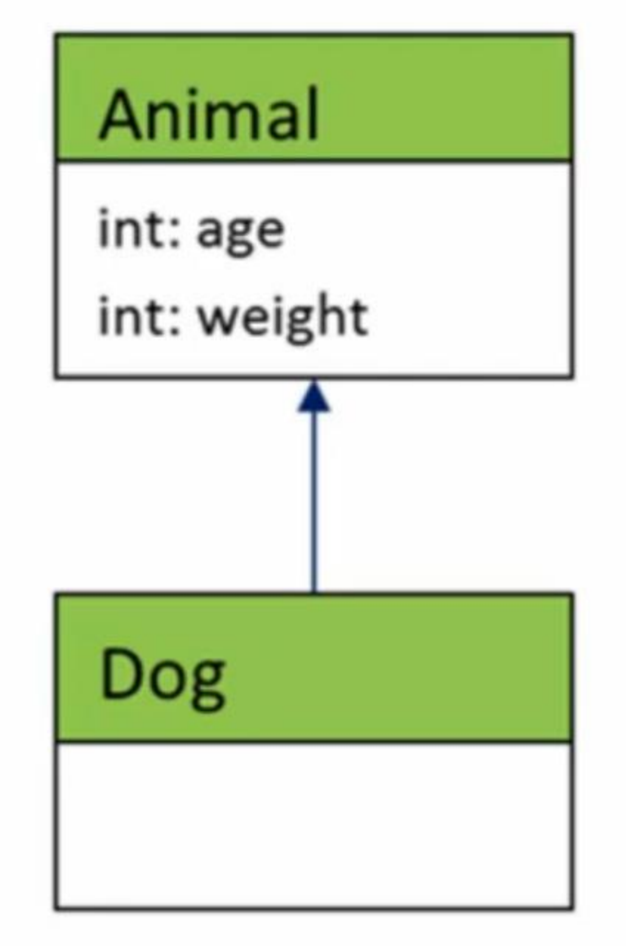
:Driver

name
employee-no
startDate
standardHourlyRate
overtimeRate
licenceType

Inheritance



Inheritance



Animal

int: age

int: weight

Dog

Message-passing

- Several objects may collaborate to fulfil each system action
- “Record CD sale” could involve:
  - A CD stock item object.
  - A sales transaction object.
  - A sales assistant object.
- These objects communicate by sending each other messages.

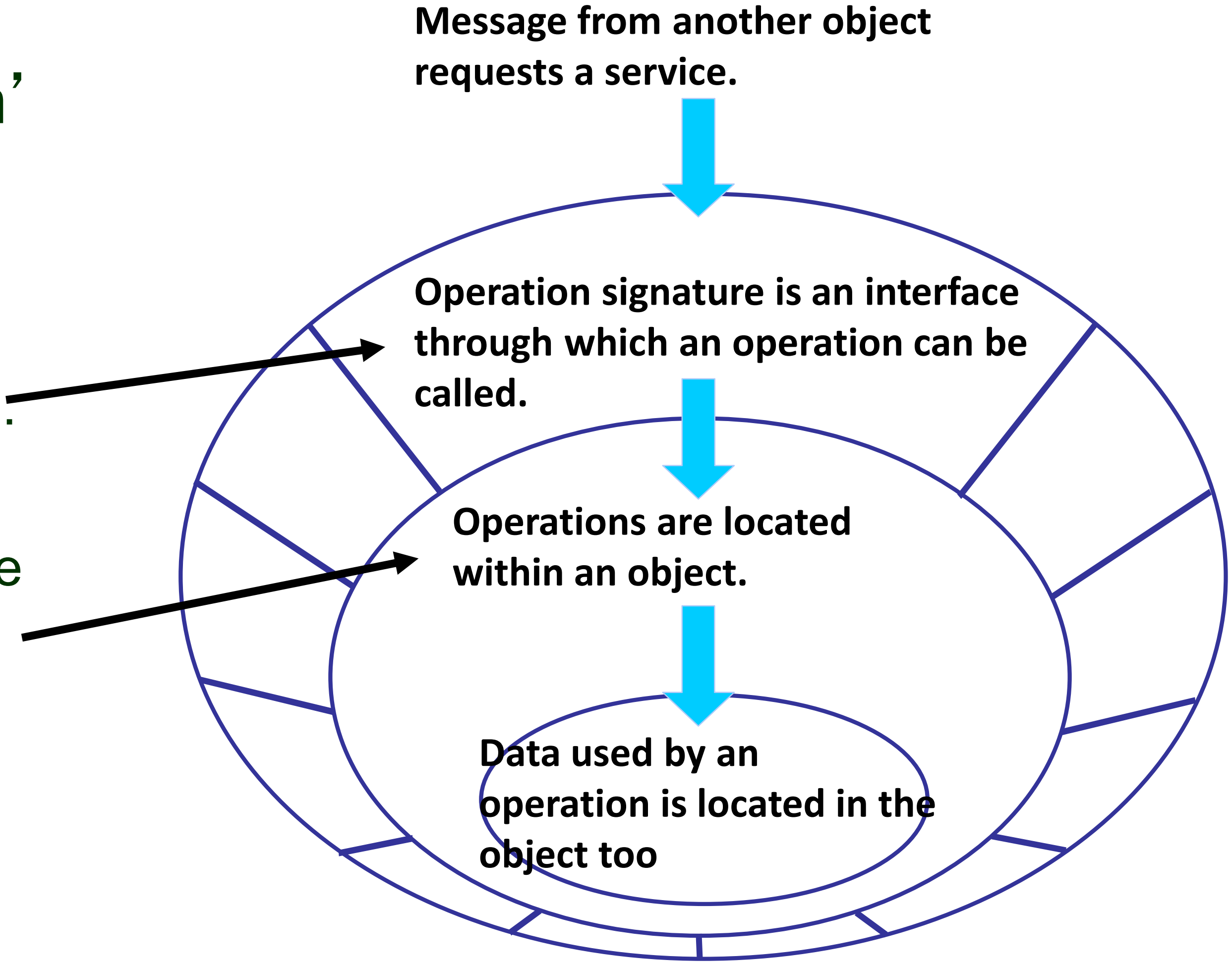
Message-passing and Encapsulation

‘Layers of an onion’ model of an object:

An outer layer of operation signatures…

…gives access to middle layer of operations…

…which access an inner core of data



Message from another object requests a service.

Operation signature is an interface through which an operation can be called.

Operations are located within an object.

Data used by an operation is located in the object too

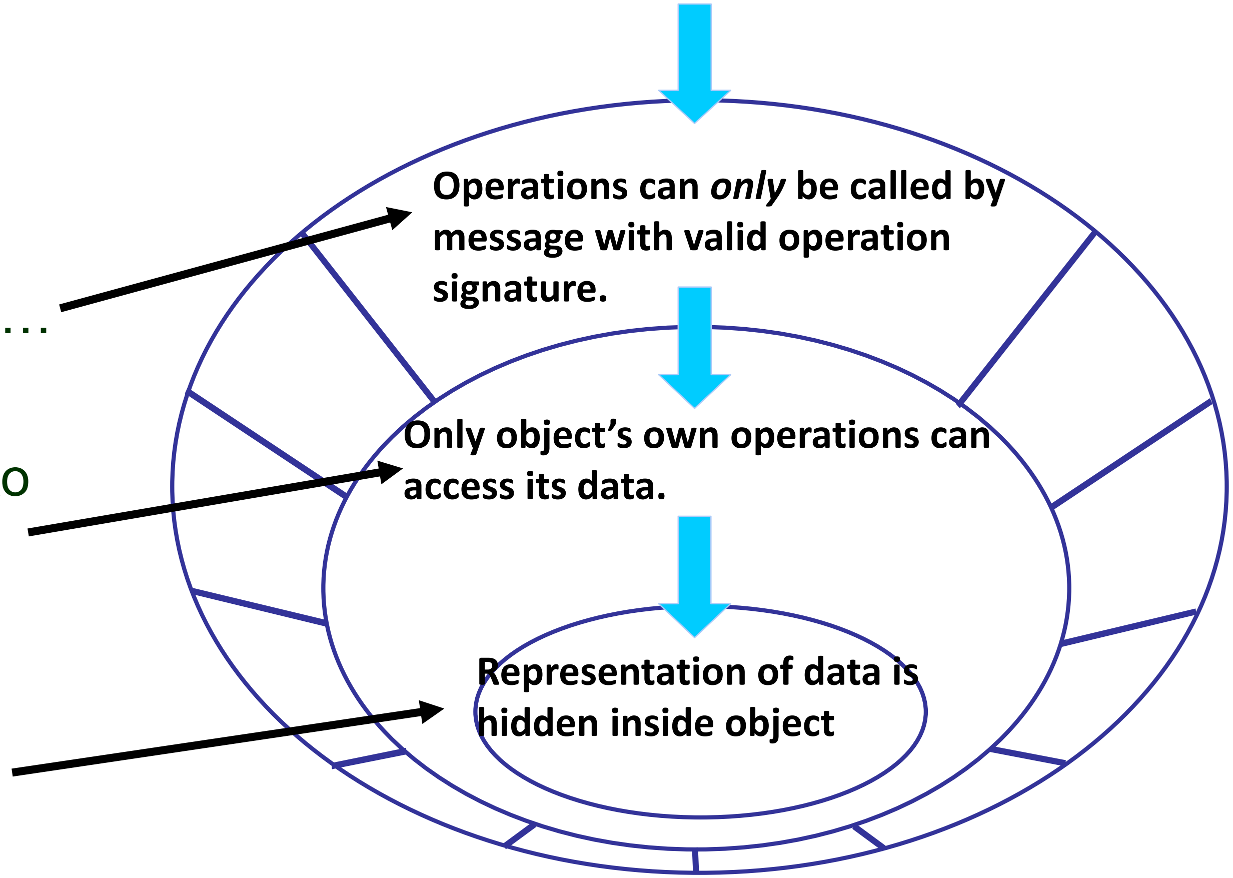
Information Hiding: a strong design principle

‘Layers of an onion’ model of an object:

Only the outer layer is visible to other objects…

…and it is the only way to access operations…

…which are the only way to access the hidden data



Message from another object requests a service.

Operations can *only* be called by message with valid operation signature.

Only object’s own operations can access its data.

Representation of data is hidden inside object

Polymorphism

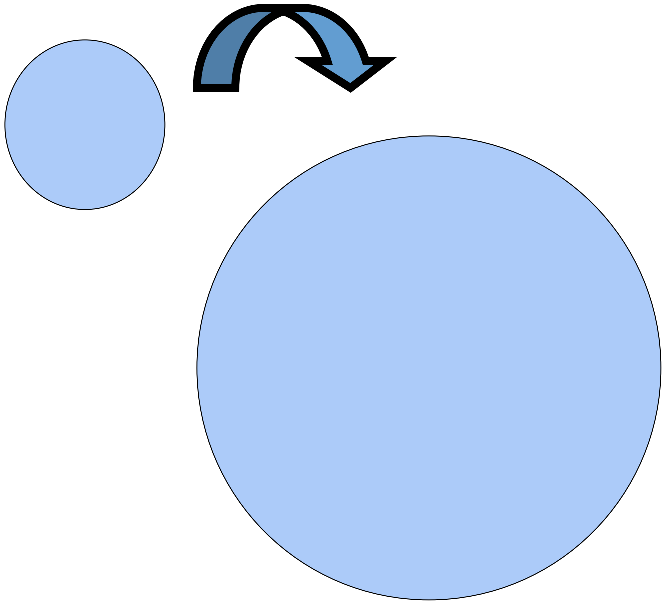
- Polymorphism allows one message to be sent to objects of different classes.

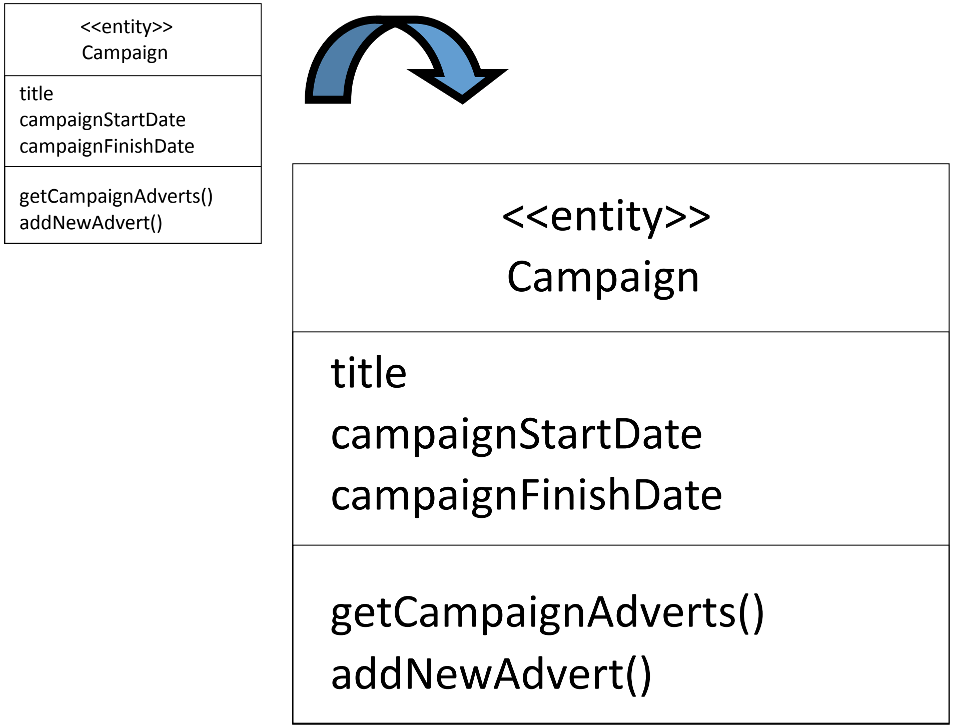
- Sending object need not know what kind of object will receive the message.

- Each receiving object knows how to respond appropriately.

- For example, a ‘resize’ operation in a graphics package.

Polymorphism in Resize Operations





<<entity>>
Campaign

title
campaignStartDate
campaignFinishDate

getCampaignAdverts()
addNewAdvert()

<<entity>>
Campaign

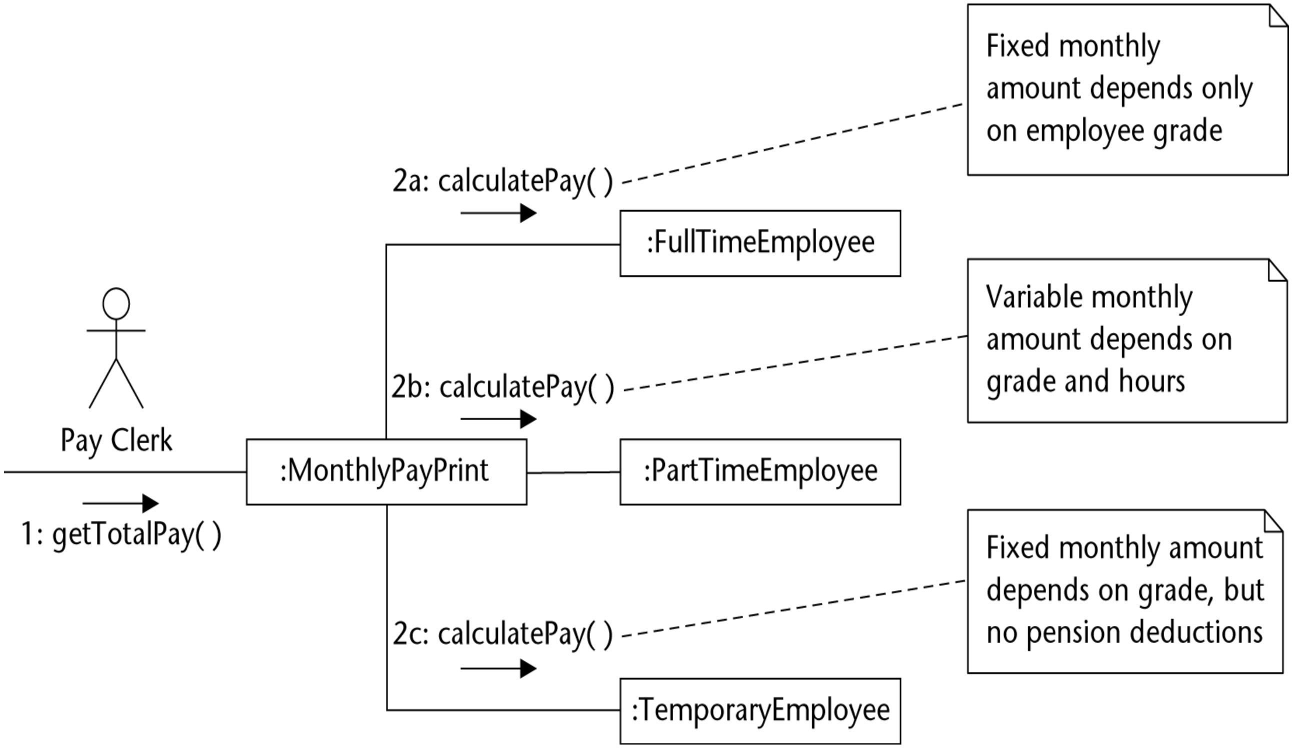
title
campaignStartDate
campaignFinishDate

getCampaignAdverts()
addNewAdvert()

Polymorphism

"Now Speak!"
"Quack"
"Woof"
"Meow"

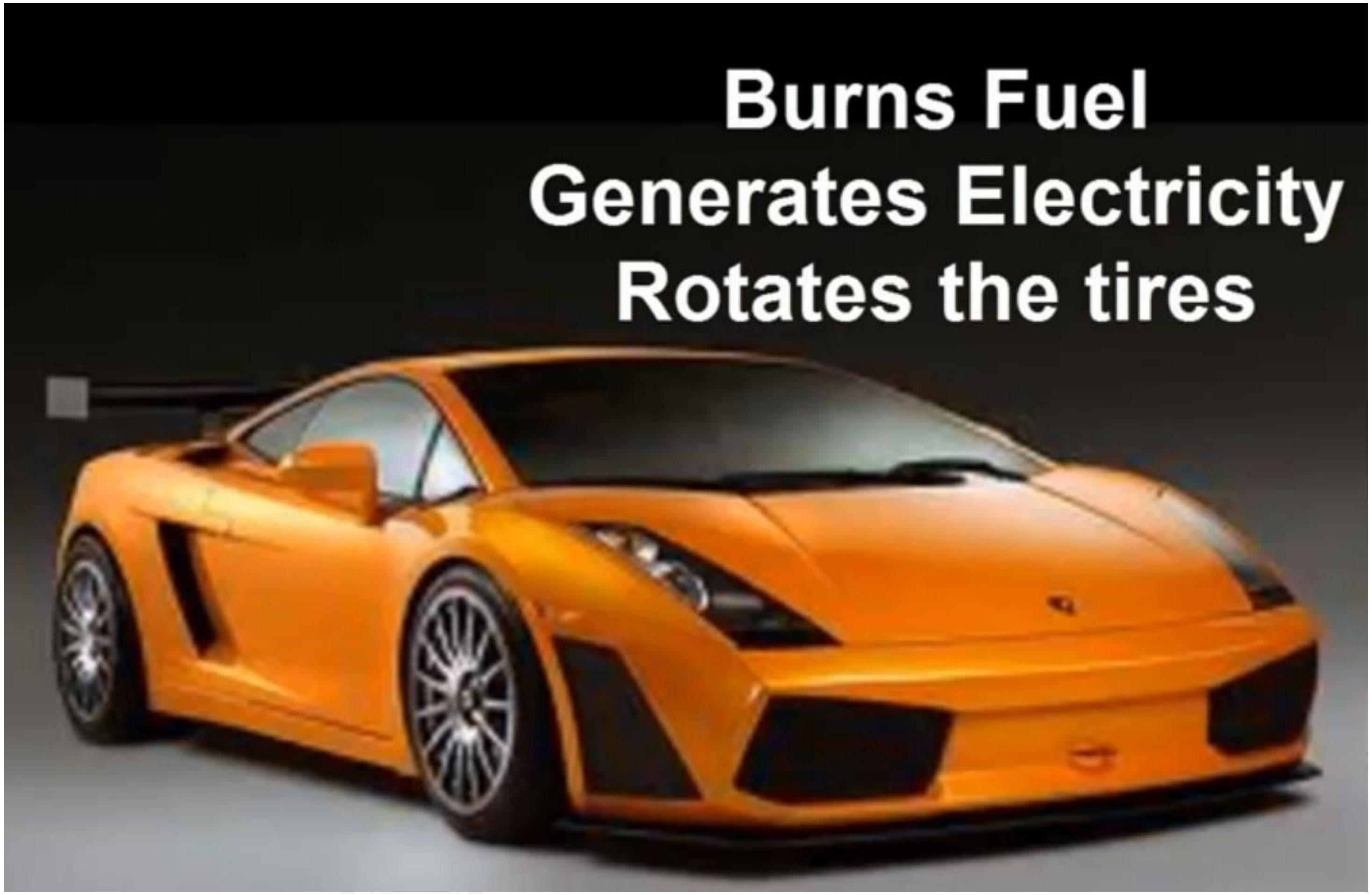
Polymorphism


Fixed monthly amount depends only on employee grade
2a: calculatePay( )
:FullTimeEmployee
Variable monthly amount depends on grade and hours
Pay Clerk
2b: calculatePay( )
:MonthlyPayPrint
:PartTimeEmployee
1: getTotalPay( )
Fixed monthly amount depends on grade, but no pension deductions
2c: calculatePay( )
:TemporaryEmployee

•There are ***different ways*** of calculating an employee's pay.
•Polymorphism allows ***a message*** to achieve the ***same*** result even when the mechanism for achieving it differs between different objects.
•A system *can* ***easily be modified*** *or* ***extended*** to include extra features

Encapsulation

Encapsulation



Advantages of Object-oriented

- Can save effort
  - Reuse of generalized components cuts work, cost and time.

- Can improve software quality
  - Encapsulation increases modularity.
  - Sub-systems less coupled to each other.
  - Better translations between analysis and design models and working code.

Structured Analysis and Design Technique (SADT)

- Diagrammatic notation for constructing a sketch for an application.

- Often considered as the conventional (traditional) way of modeling.

- Two types:
  - Process model.
  - Data model.

Structured Analysis and Design Technique (SADT)

- A *process model* is a graphical way of representing how a business system should operate
  - It illustrates the processes or activities that are performed and how data move among them.

- A *data model* is a formal way of representing the data that are used and created by a business system
  - it illustrates people, places, or things about which information is captured and how they are related to each other.

Diagram in SADT

- SADT model uses:
    - boxes to represent entities and activities.
    - variety of arrows to relate boxes.
    - boxes and arrows have an associated (informal) semantics; users are aided by box and arrow labels, other informal documentation.

Data Flow Diagram

- Data flow diagramming is a technique that diagrams the business processes and the data that pass among them.

- The focus is mainly on the processes or activities that are performed.

- Presents how the data created and used by processes are organized

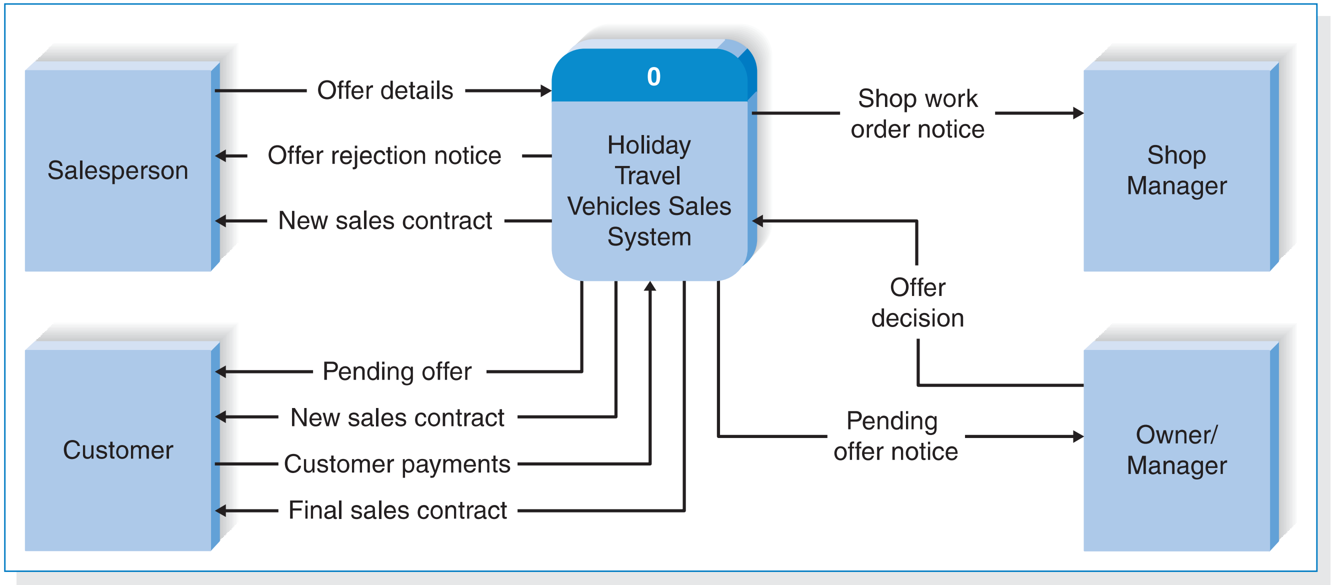
Using DFD

- Most business processes are too complex to be explained in one DFD.

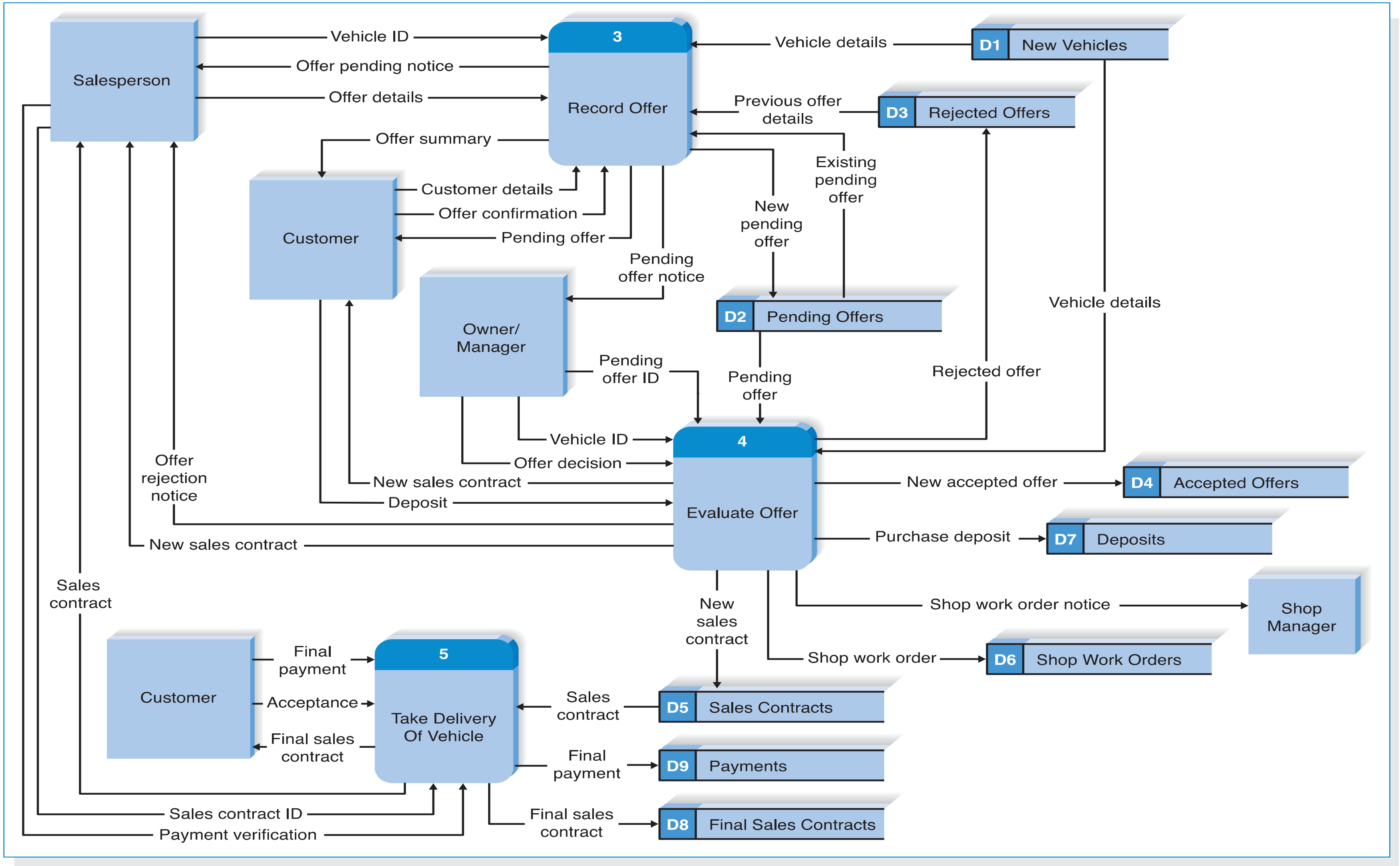
- The principle in process modeling with DFDs is the decomposition of the business process into a hierarchy of DFDs, with each level down the hierarchy representing less scope but more detail.

- The first DFD (Context Diagram) provides a summary of the overall system
    - with additional DFDs (Level 0, 1 ..) providing more and more detail about each part of the overall business process.

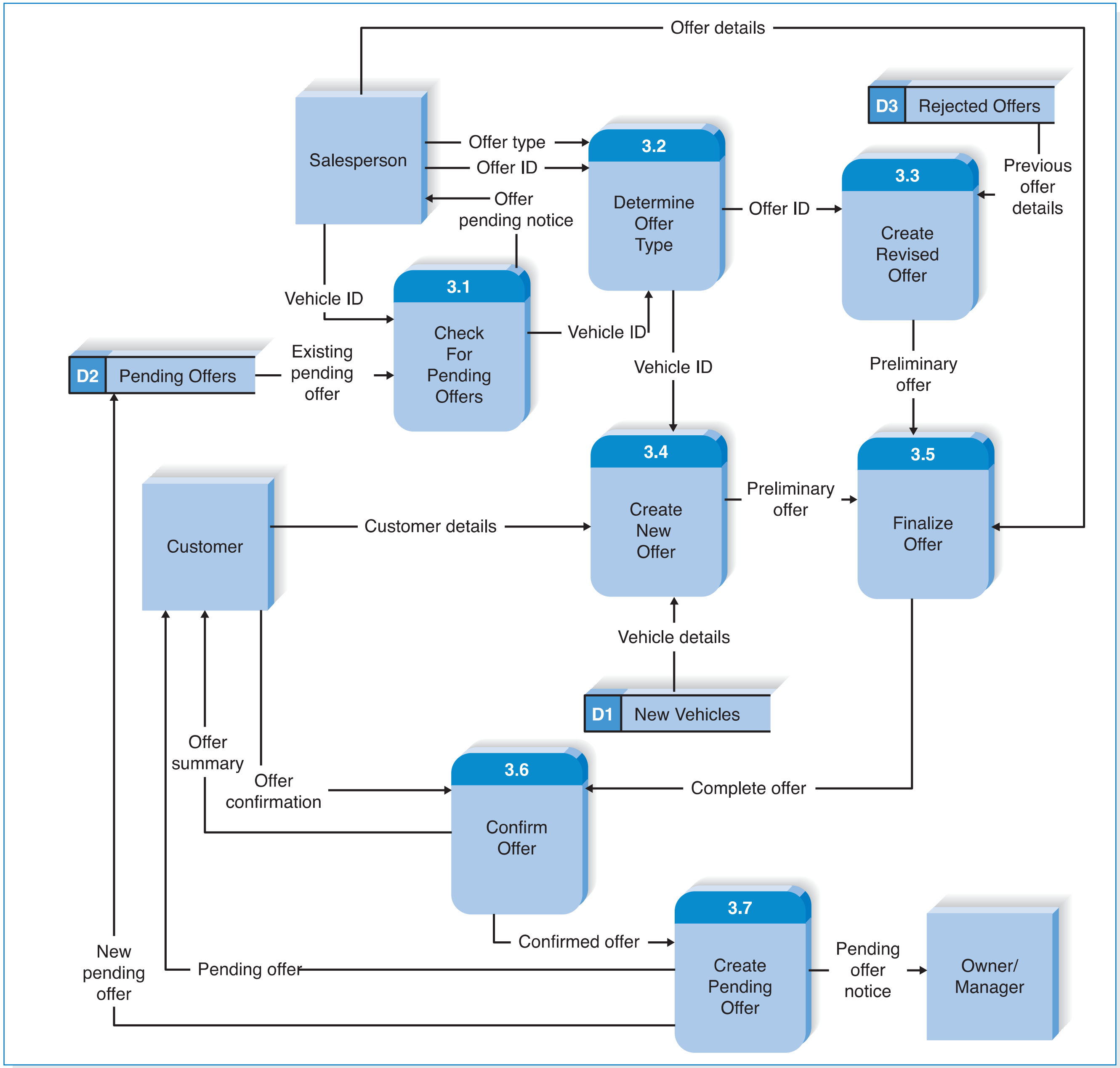
Holiday Travel Vehicle Context Diagram



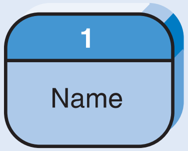
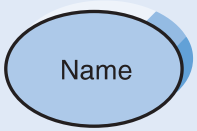
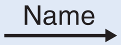
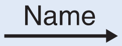
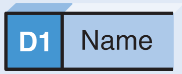
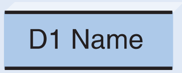
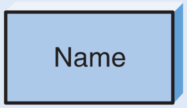
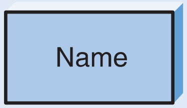
Holiday Travel Vehicle Level 0 DFD



Holiday Travel Vehicle Level 1 DFD



Elements of DFD

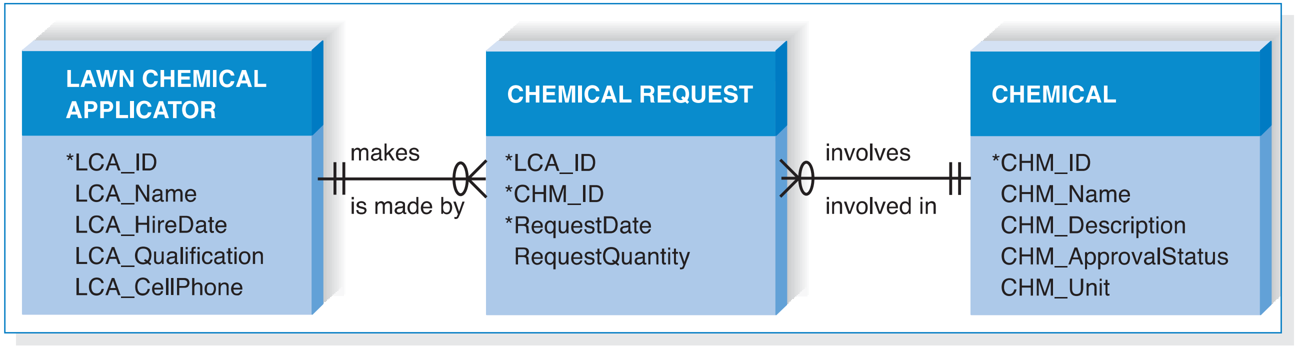
Data Flow Diagram Element	Typical Computer-Aided Software Engineering Fields	Gane and Sarson Symbol	DeMarco and Yourdon Symbol
Every *process* has a number a name (verb phase) a description at least one output data flow at least one input data flow	Label (name) Type (process) Description (what is it) Process number Process description (structured English) Notes		
Every *data flow* has a name (a noun) a description one or more connections to a process	Label (name) Type (flow) Description Alias (another name) Composition (description of data elements) Notes		
Every *data store* has a number a name (a noun) a description one or more input data flows one or more output data flows	Label (name) Type (store) Description Alias (another name) Composition (description of data elements) Notes		
Every *external entity* has a name (a noun) a description	Label (name) Type (entity) Description Alias (another name) Entity description Notes		

Entity Relationship Diagram (ERD)

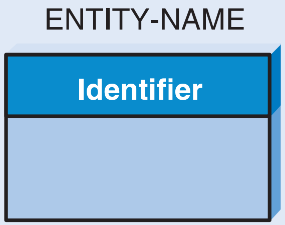
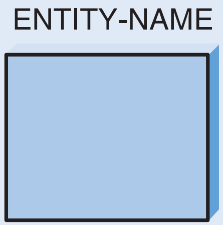
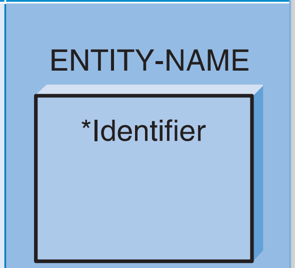
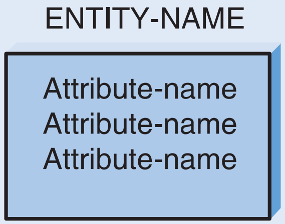
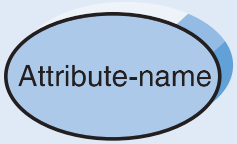
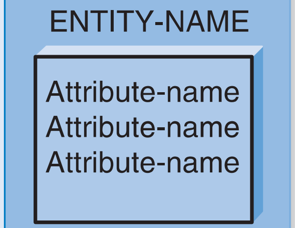
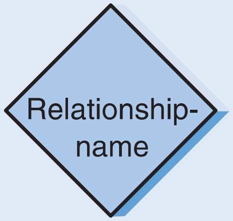
- Entity relationship diagramming is a graphic drawing technique that shows all the data components of a business system.

- An *entity relationship diagram* (*ERD*) is a picture which shows the information that is created, stored, and used by a business system.

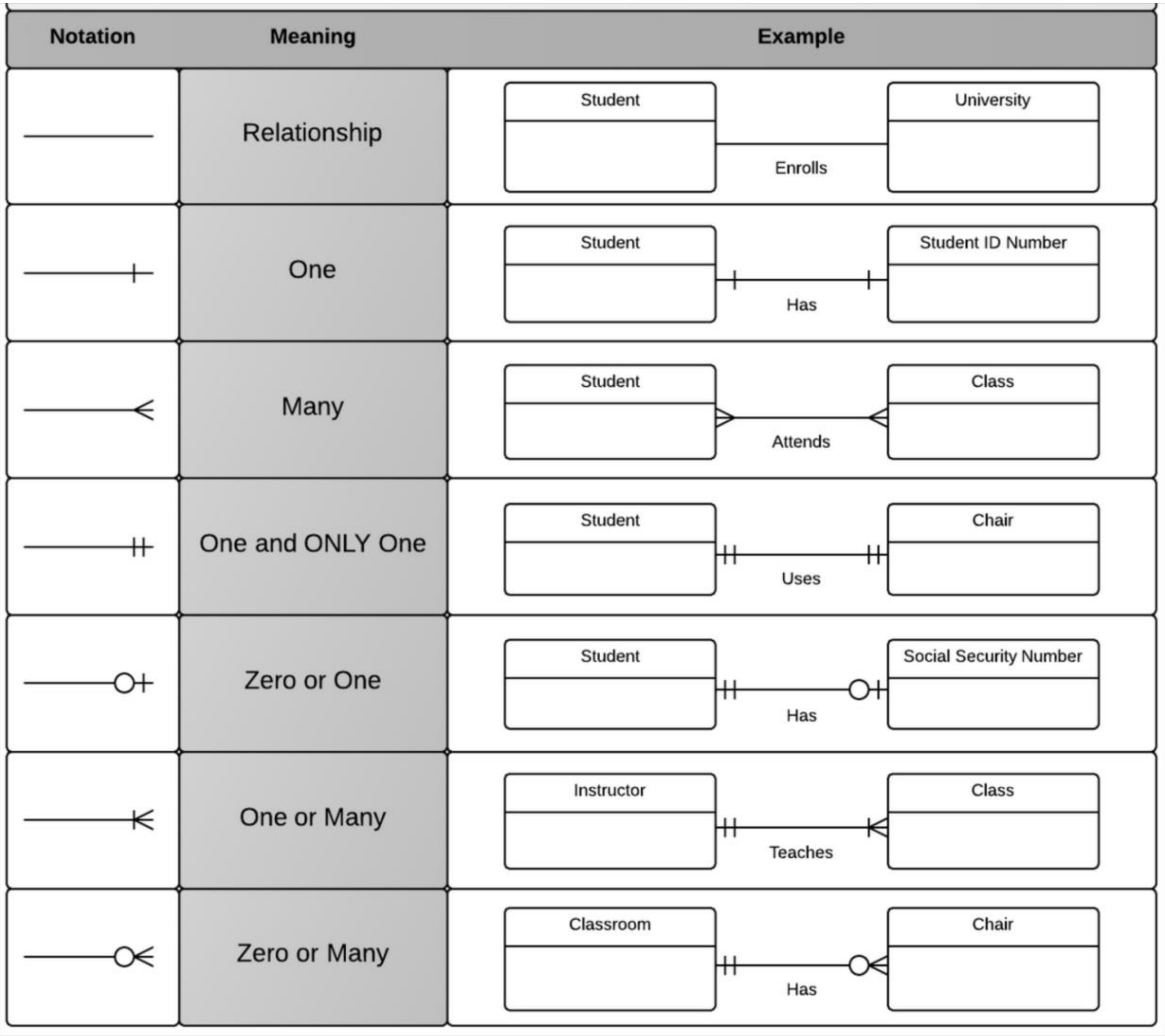
Chemical Request ERD



Data modelling symbol sets

	IDEF1X	Chen	Crow's Foot
An ENTITY ✓ is a person, place, or thing. ✓ has a singular name spelled in all capital letters. ✓ has an identifier. ✓ should contain more than one instance of data.	 ENTITY-NAME Identifier	 ENTITY-NAME	 ENTITY-NAME *Identifier
An ATTRIBUTE ✓ is a property of an entity. ✓ should be used by at least one business process. ✓ is broken down to its most useful level of detail.	 ENTITY-NAME Attribute-name Attribute-name Attribute-name	 Attribute-name	 ENTITY-NAME Attribute-name Attribute-name Attribute-name
A RELATIONSHIP ✓ shows the association between two entities. ✓ has a parent entity and a child entity. ✓ is described with a verb phrase. ✓ has cardinality (1 : 1, 1 : N, or M : N). ✓ has modality (null, not null). ✓ is dependent or independent.	Relationship-name	 Relationship-name	Relationship-name

ERD relationship – Crow's Foot



Notation	Meaning	Example
	Relationship	Student Enrolls University
	One	Student Has Student ID Number
	Many	Student Attends Class
	One and ONLY One	Student Uses Chair
	Zero or One	Student Has Social Security Number
	One or Many	Instructor Teaches Class
	Zero or Many	Classroom Has Chair

Object oriented vs. SADT

Criteria	Structured	Object-Oriented
Process Methodology	Traditional SDLC such as Waterfall	Iterative/IncrementalFocus
Focus	Processes	Objects
Risk	High	Low
Reuse	Low	High
Suitable for	Well-defined projects with stable user requirements	Risky large projects with changing user requirements
Analysis	Structuring Requirements • Data flow diagrams • Structured English • Decision Table/Tree • Entity Relationship (ER) Analysis	Requirements Engineering • Use case Model (find use cases, flow of events, activity diagram) • Object Model ○ Find classes and class relations ○ Object interactions ○ Object to ER mapping
Design	• Database design (DB normalization) • GUI Design (Forms & reports)	• Physical Database design • Design elements (System architecture, classes, components), GUI)

Key points

- A model an abstract representation of something real or imaginary.

- A diagram illustrates some aspect of a system while a model provides a complete view of a system at a particular stage and from a particular perspective.

- The UML model is define using diagrams that are defined in the UML specification.

Key points

- The fundamental concepts of object-oriented includes
  - Object, class, instance
  - Generalization and specialization
  - Message-passing and polymorphism

- Structured Analysis and Design Technique (SADT) is a traditional approach to modelling software.

- SADT uses two types of model; process and data model.

References

- Booch, Rumbaugh and Jacobson (1999)
- Bennett, Skelton and Lunn (2005)

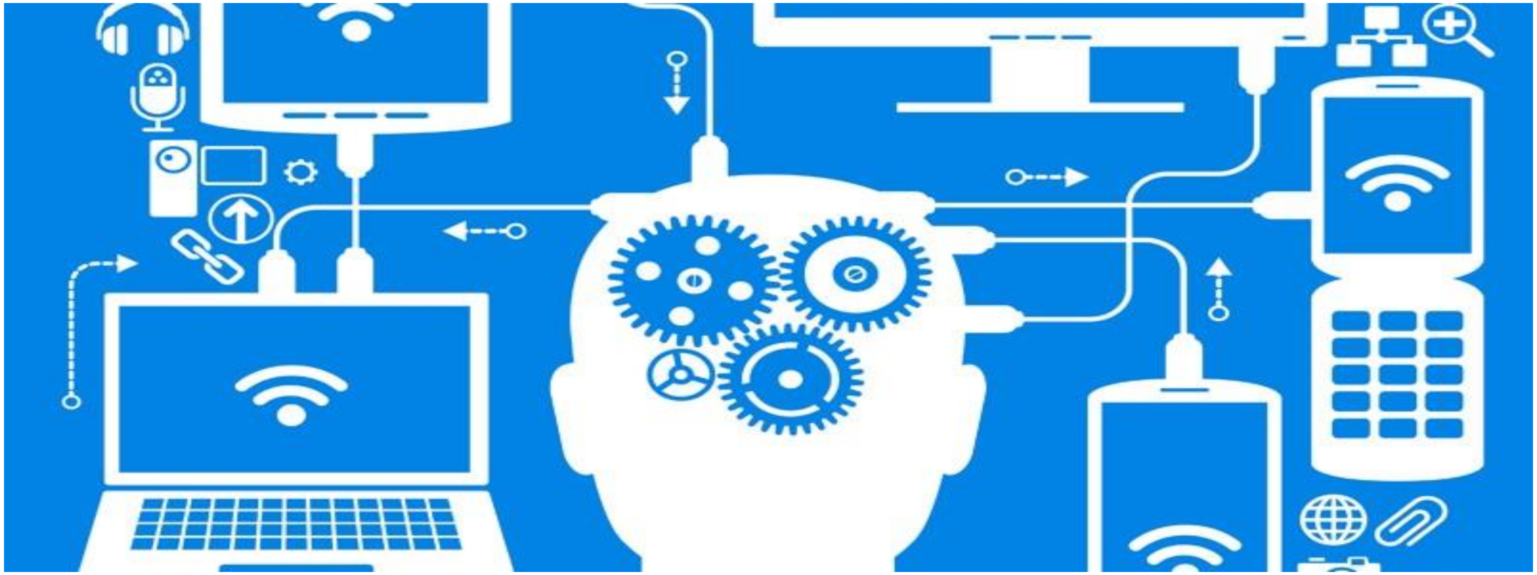
(For full bibliographic details, see Bennett, McRobb and Farmer)

References

- Coad and Yourdon (1990)
- Booch (1994)
- OMG (2009)

(For full bibliographic details, see Bennett, McRobb and Farmer)

In the next lecture..



Lecture 4: Requirements Analysis – UML Use Case Diagram