CHAPTER 2: INTRODUCTION TO OBJECT ORIENTATION

- An Introduction To Object Modeling
- System Concept for Object Modeling
- The Overall View Components of UML Diagram

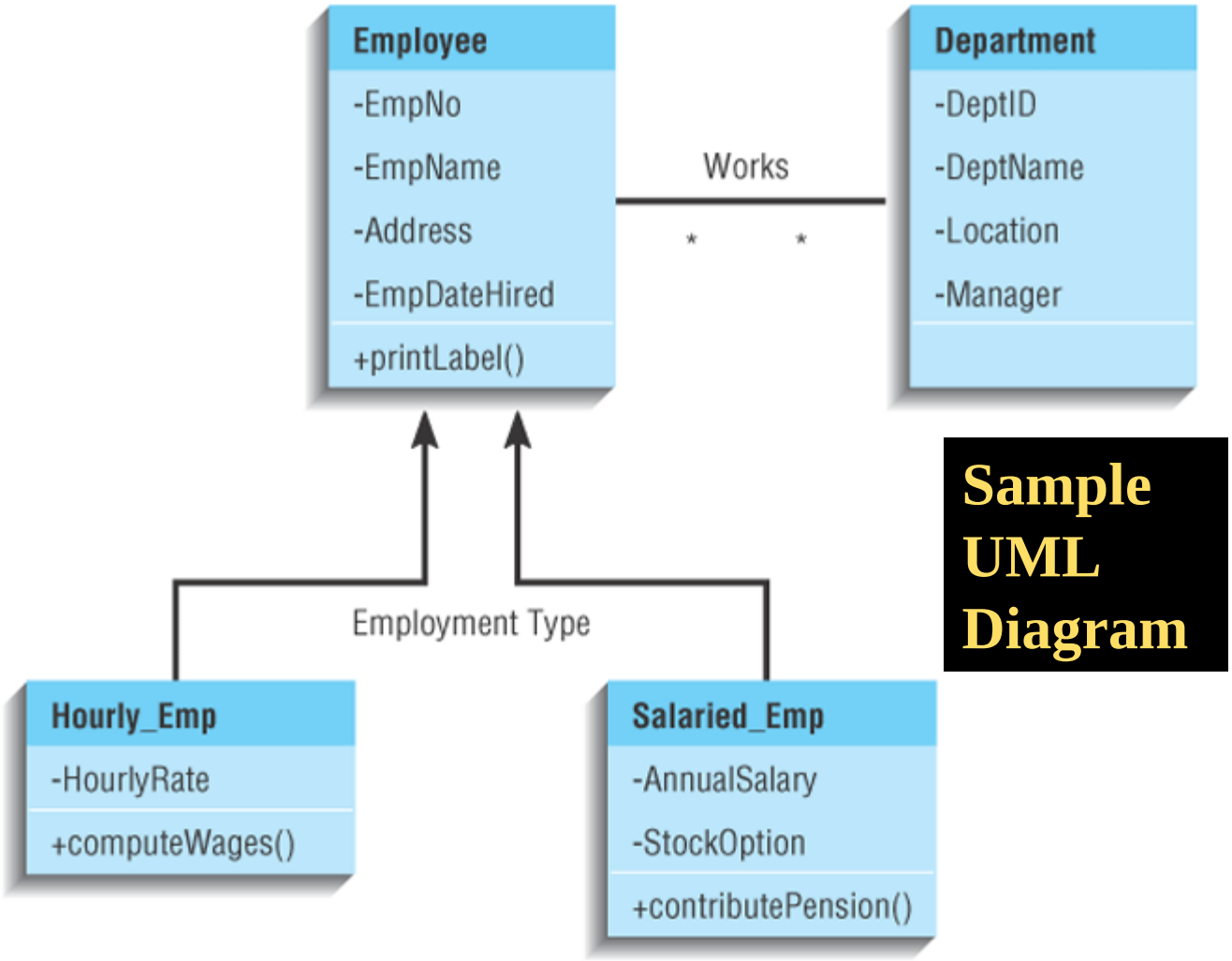
CHAPTER OBJECTIVES

- After studying this chapter you should be able to:
  - Define an object.
  - Understand the terms *class*, *attribute*, and *operations*.
  - Explain generalization, polymorphism, and inheritance.
  - Define association.
  - Describe modeling and the Unified Modeling Language.

UNIFIED MODELING LANGUAGE (UML)

- A standard notation for representing object-oriented systems
- Boxes represent classes, components, packages, objects
  - Containing attributes and operations
  - Provide interfaces to external entities
- Lines represent generalization and other relationships

Figure 2.1 General Class Employee, with Two Specific Classes, Hourly Employee and Salaried Employee, with an Association with the Class Department



Sample UML Diagram

THE UNIFIED MODELLING LANGUAGE

- During the 90s there were several disparate modelling methods for object oriented programming.
  - The used the same concepts with different notations.
  - This produced lots of confusion among designers and programmers.
  - "The war of the methods"
- In 1994, Booch, Rumbaugh (OMT) y Jacobson (Objectory) decided to unify their methods and created:

  Unified **M**odeling **L**anguage (UML)
- This was a standardisation method promoted by OMG

OBJECT ORIENTED ANALYSIS METHODS

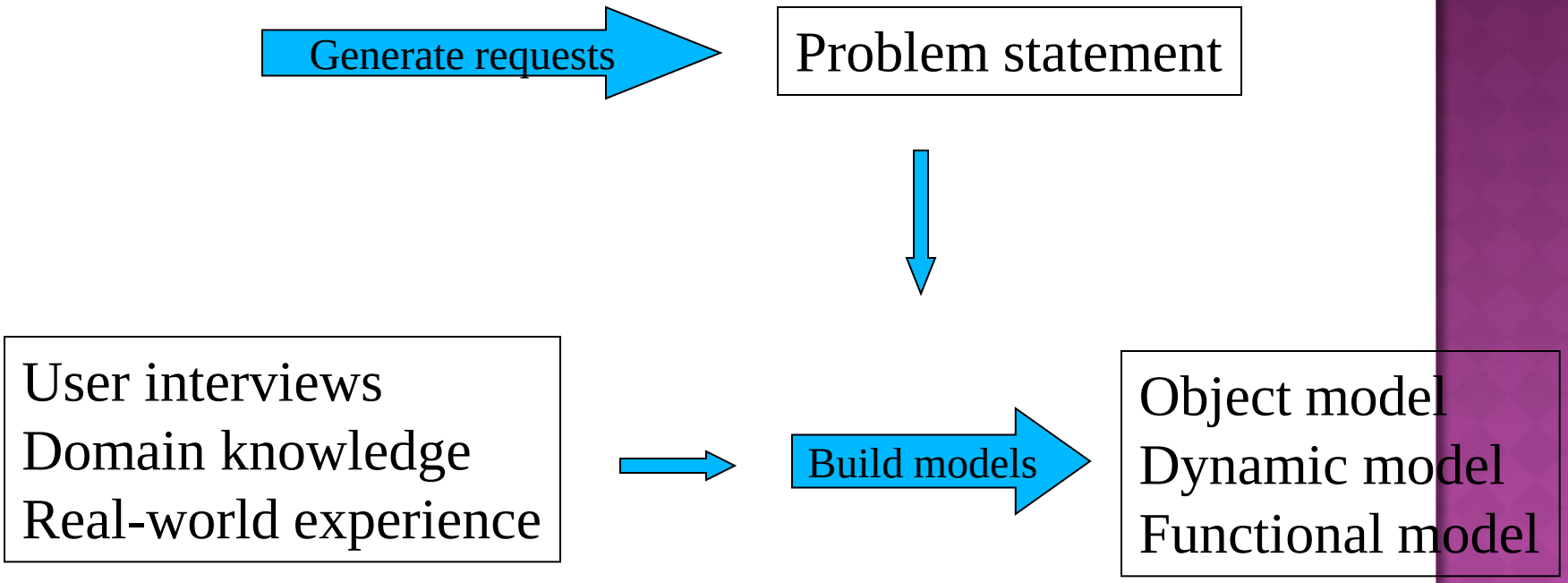
Booch
OMT
Jacobson
Shlaer-Mellor
Wirfs-Broks
Fusion
Catalysis

Coad/Yourdon
Champeaux
Martin/Odell
OOram
BON
Open

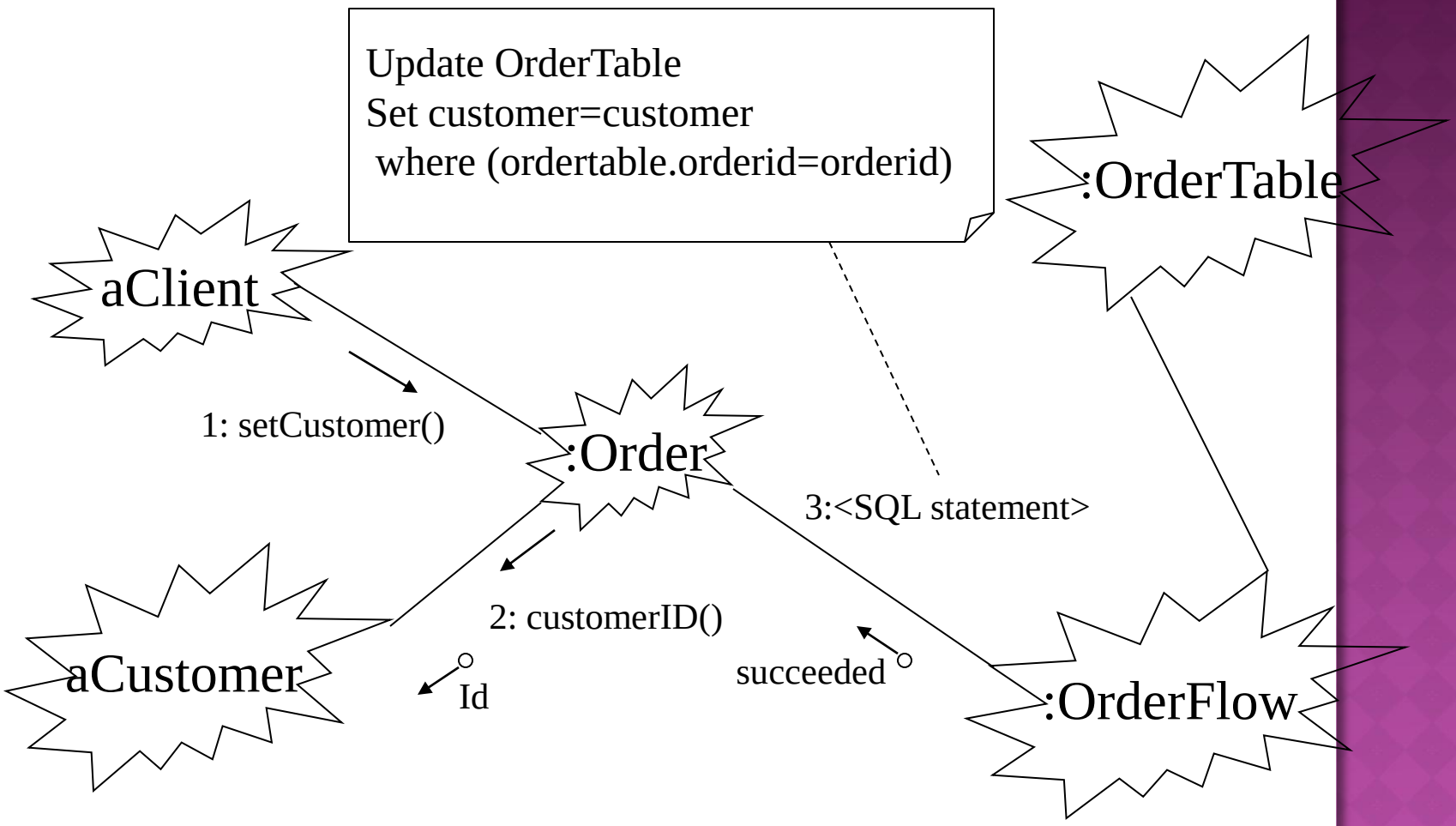
And many many more!

Jacobson's View of System Development

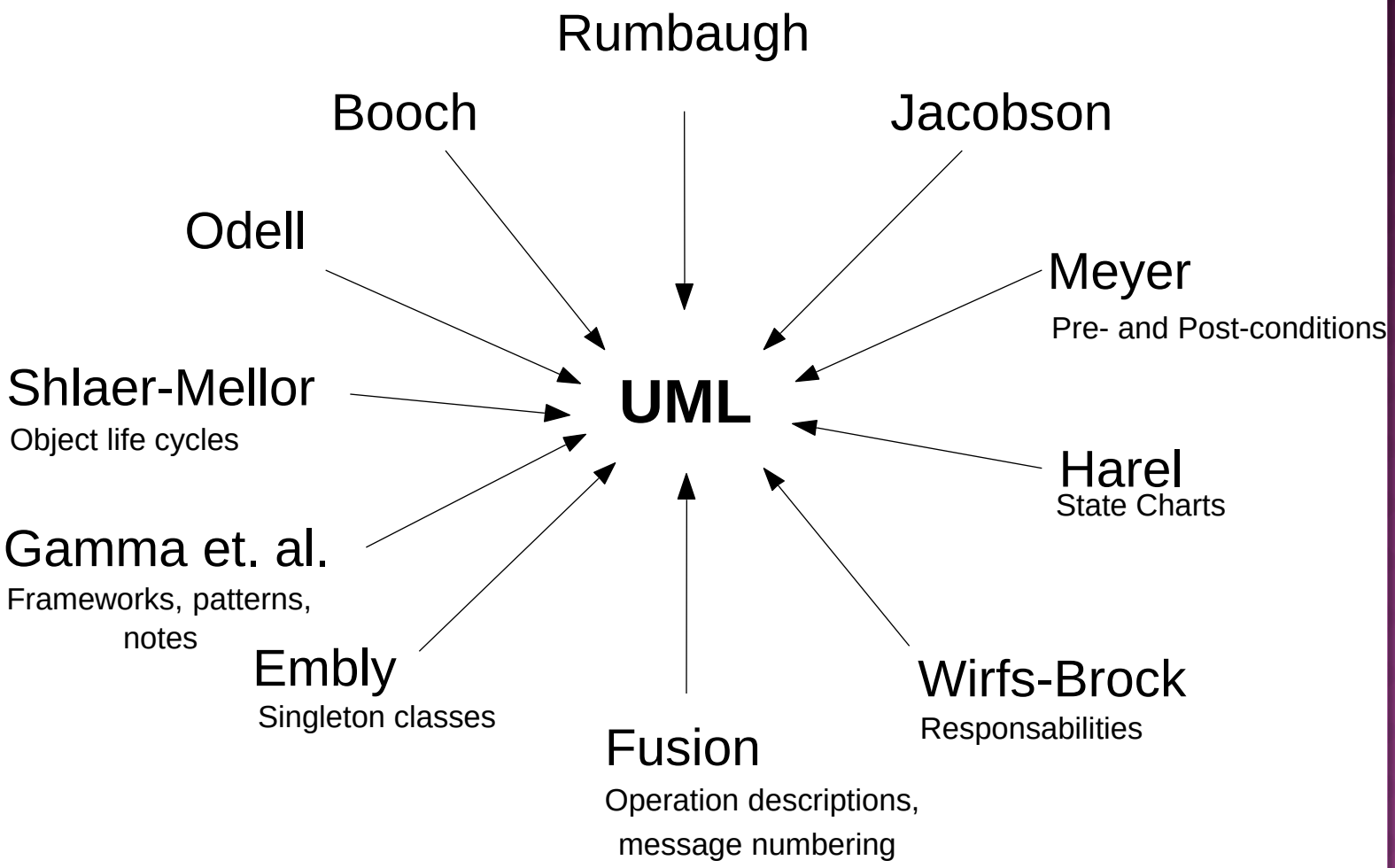
Rumbaugh's View of System Analysis



Booch's View of System Design



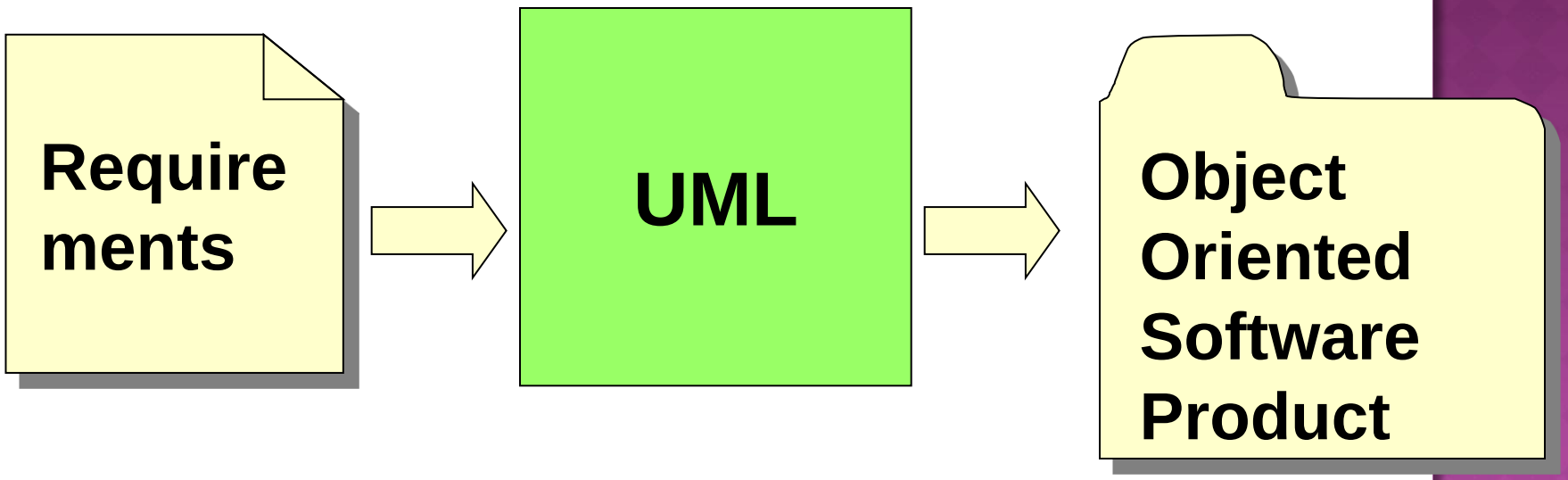
UML ENCOMPASSES VARIOUS METHODS



ADVANTAGES OF THE UNIFICATION

- Has all the strong points of each method
- Promotes new ways of improving designs/analysis
- Promotes stability in the software industry
- Simplifies users life!

FROM THE SPECIFICATION TO THE SOLUTION



UML AND MODELLING

UML is a language to visualize, specify, build and document the artefacts (models) of a system which involves a large number (or size) of object oriented software components.

- UML is a notation, not a design process
- Various design processes are being design with UML in mind.
- E.g. Rational created *RUP*, which is a "*unified process*".
- UML could be used for systems which are not necessarily software based.

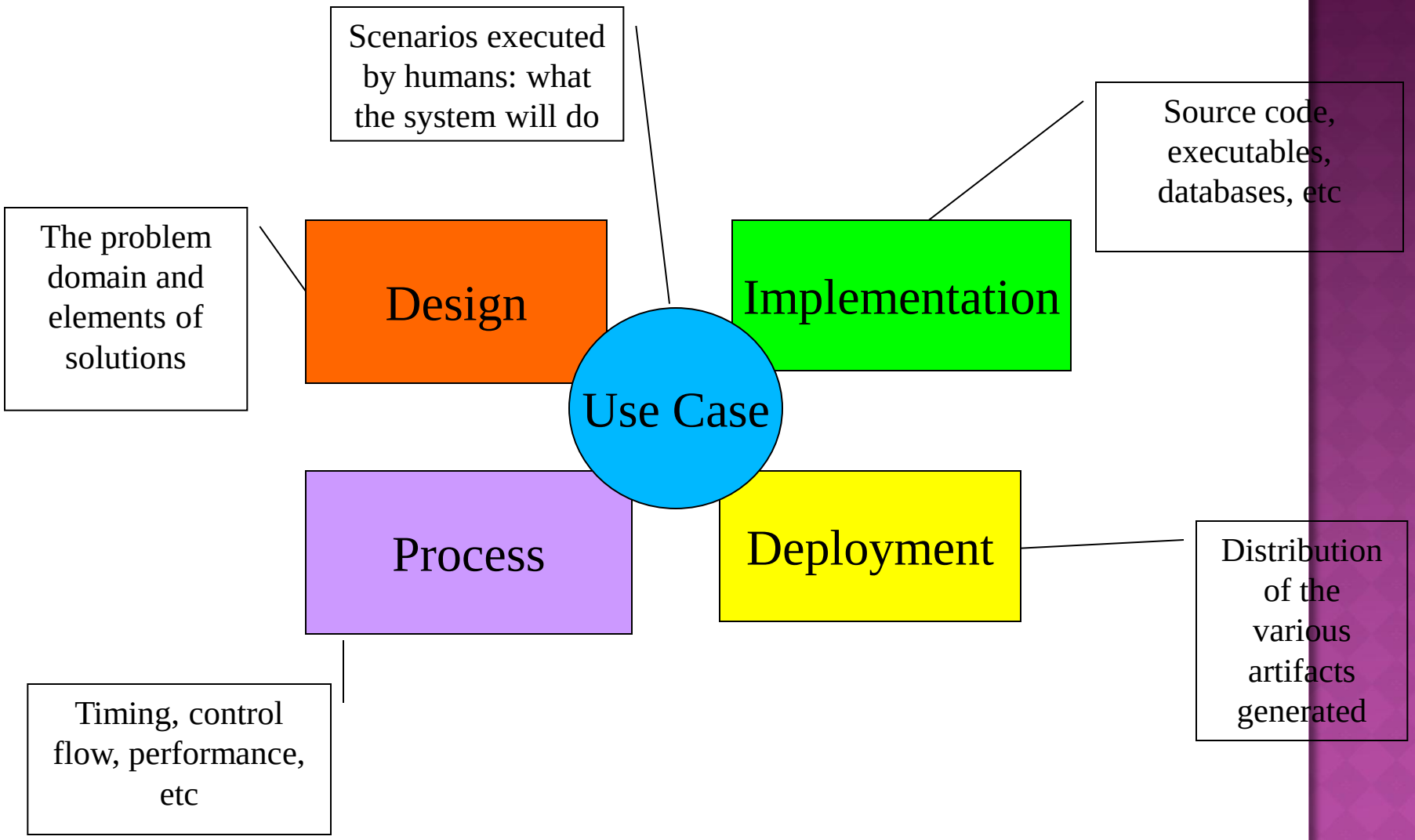
WHAT IS UML USEFUL FOR?

- Allows to:
  - Specify all the analysis, design and implementation decisions.
  - Build models which are precise, unambiguous and complete
  - Documents all the development steps (e.g. requirements, architecture, tests, versioning, etc..)

GOALS

- Provide users a expressive, visual language to enable sharing of meaningful models.
- Enables developers to compare models before implementation phase.
- Independent of programming language and development process.
- Encourage growth of Object Oriented Tools market.

UML 5 Views of a Software System



UML CHARACTERISTICS

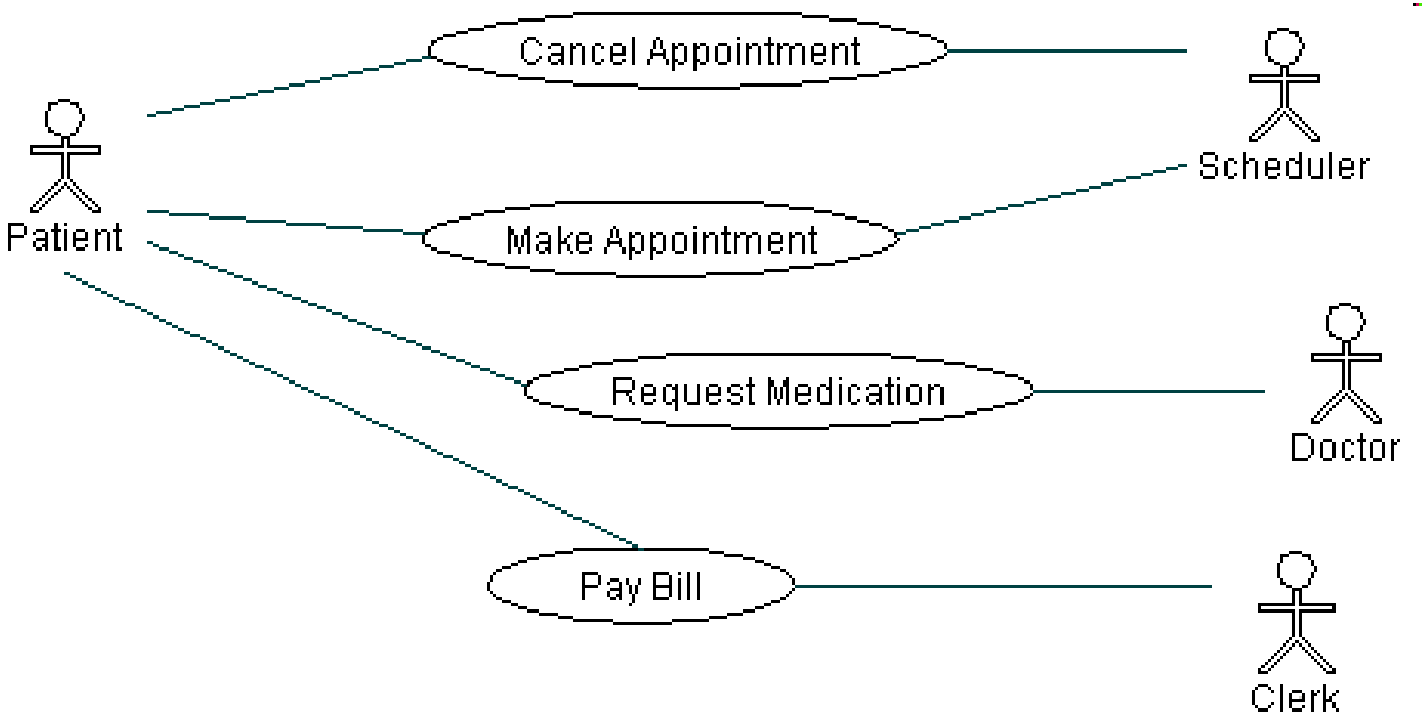
- UML is independent of the implementation language.
- UML could be implemented in various languages
- We will focus on UML tied to java

UML DIAGRAMS

- **UML defines several types of diagrams:**
  - **Use Case**
  - **Classes**
  - **Packages**
  - **Objects**
  - **Interaction**
    - Sequence
    - Collaboration
  - **States**
  - **Activities**
  - **Components**
  - **Deployment**

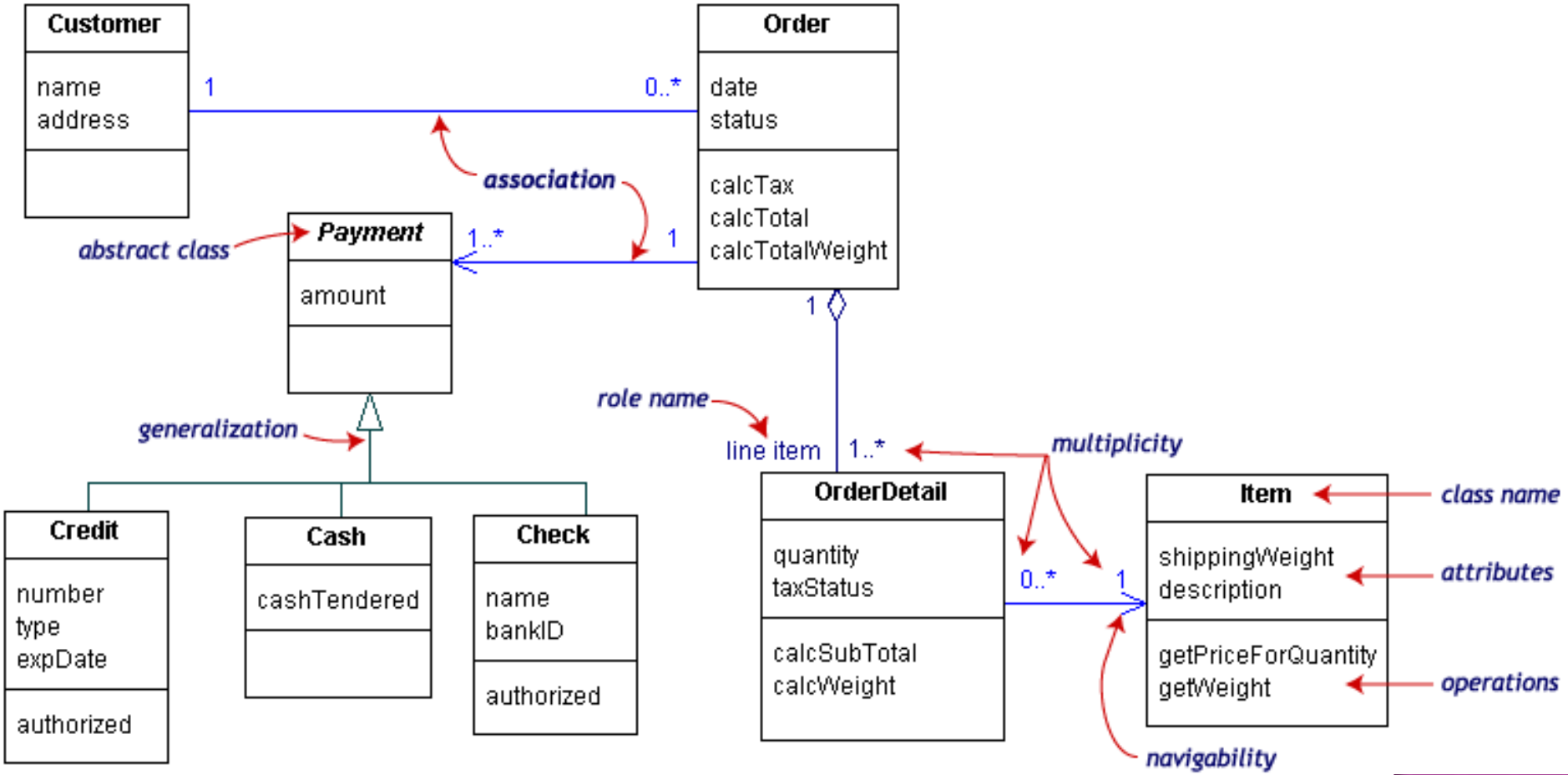
UML also provides mechanisms to extend itself!

USE CASE DIAGRAMS



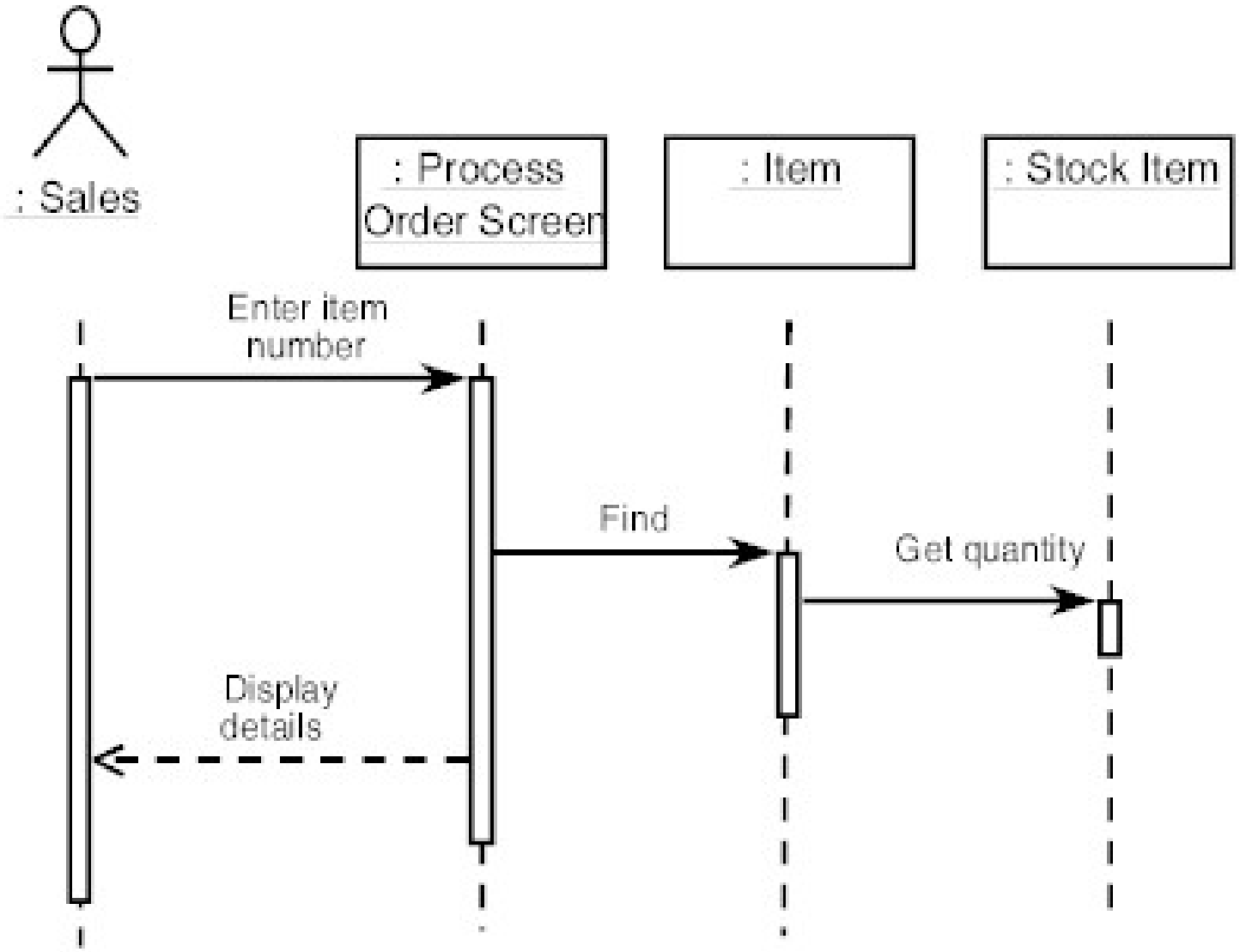
Use Case diagrams describe what a system does. HEASSISTANT simply had 1 actor, but generally, we have multiple actors.

CLASS DIAGRAMS

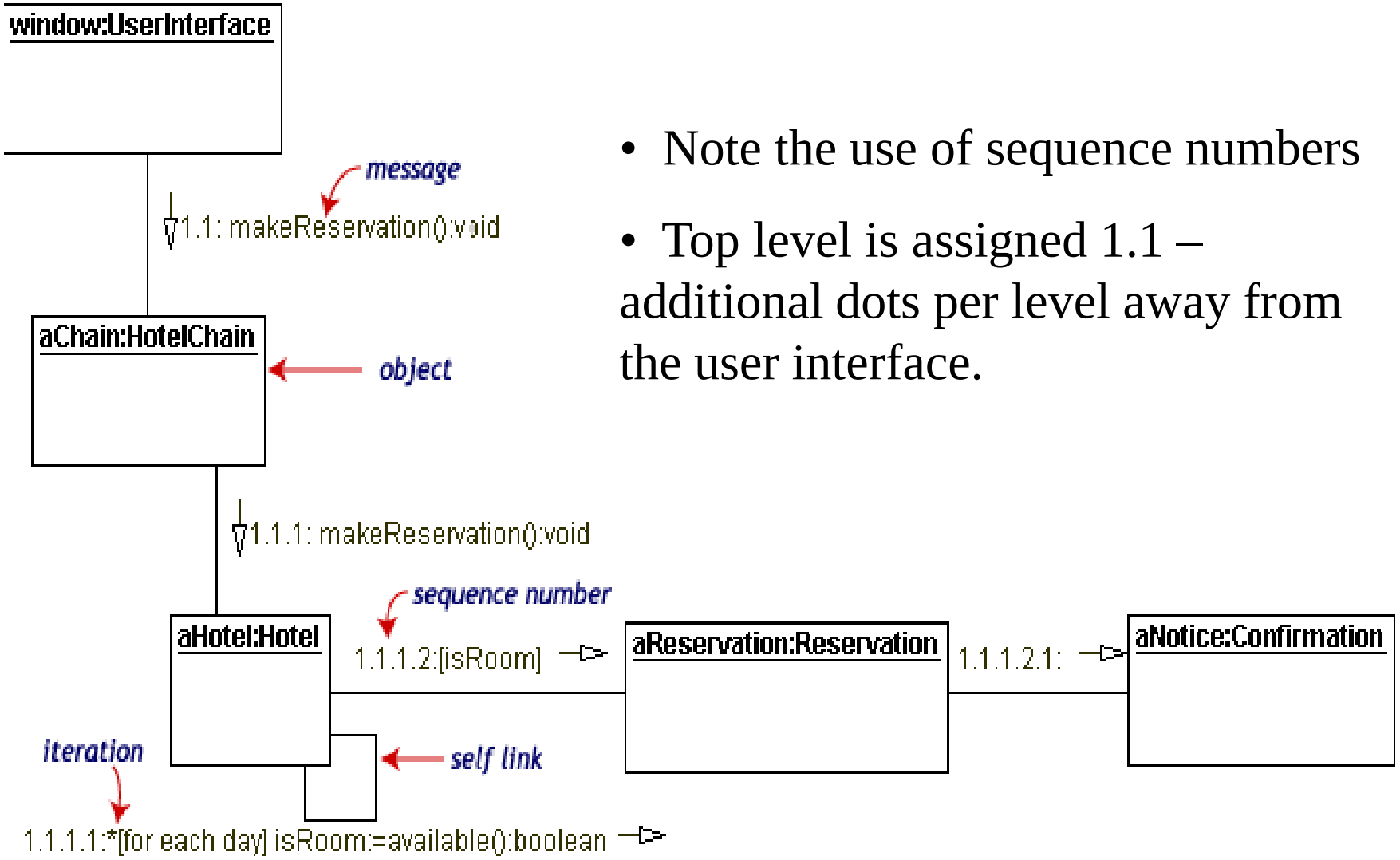


Note advanced additional features such as abstract classes, generalization (inheritance), aggregation (orderdetails make up order)

SEQUENCE DIAGRAMS

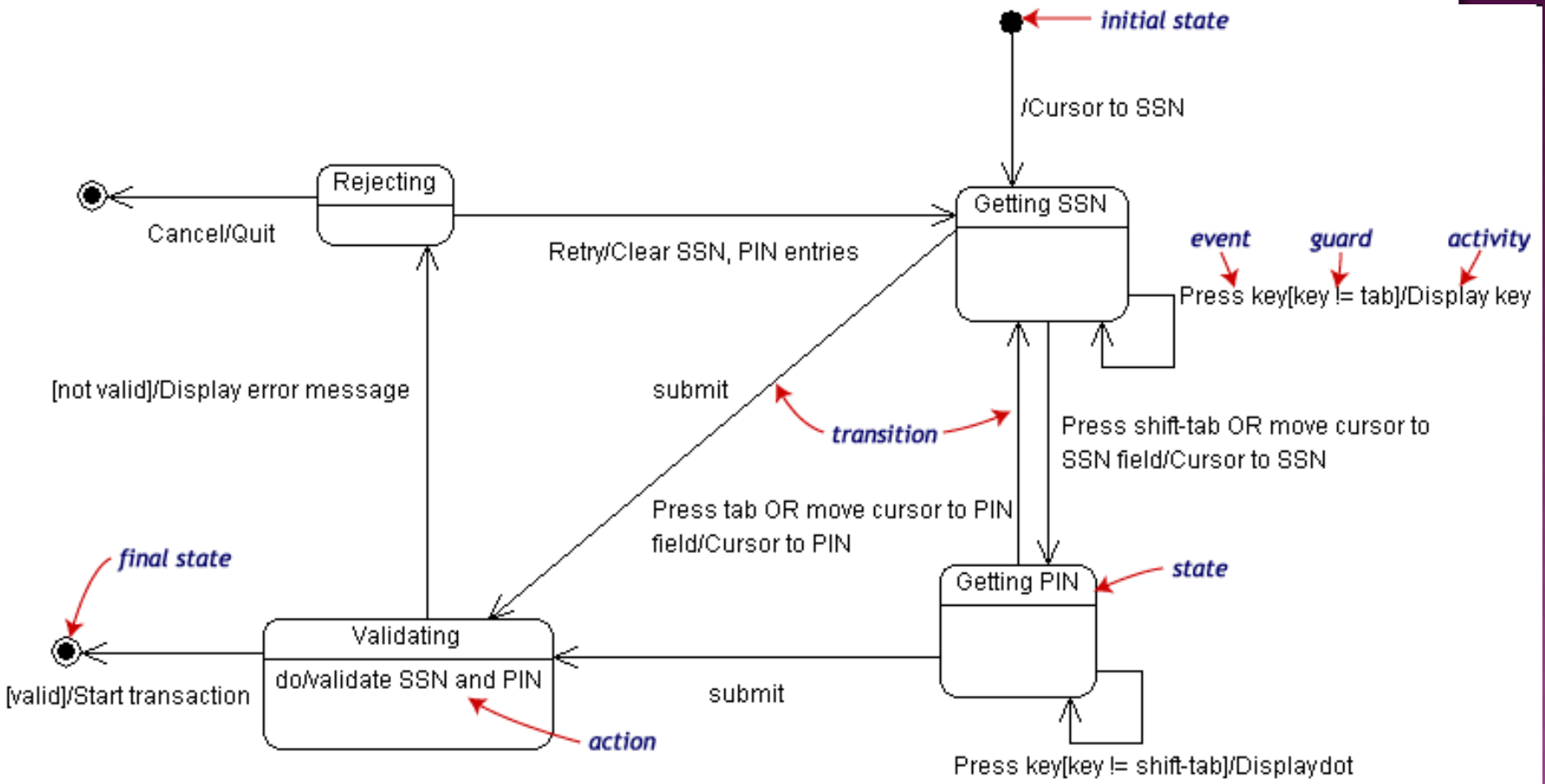


COLLABORATION DIAGRAMS



- Note the use of sequence numbers
- Top level is assigned 1.1 – additional dots per level away from the user interface.

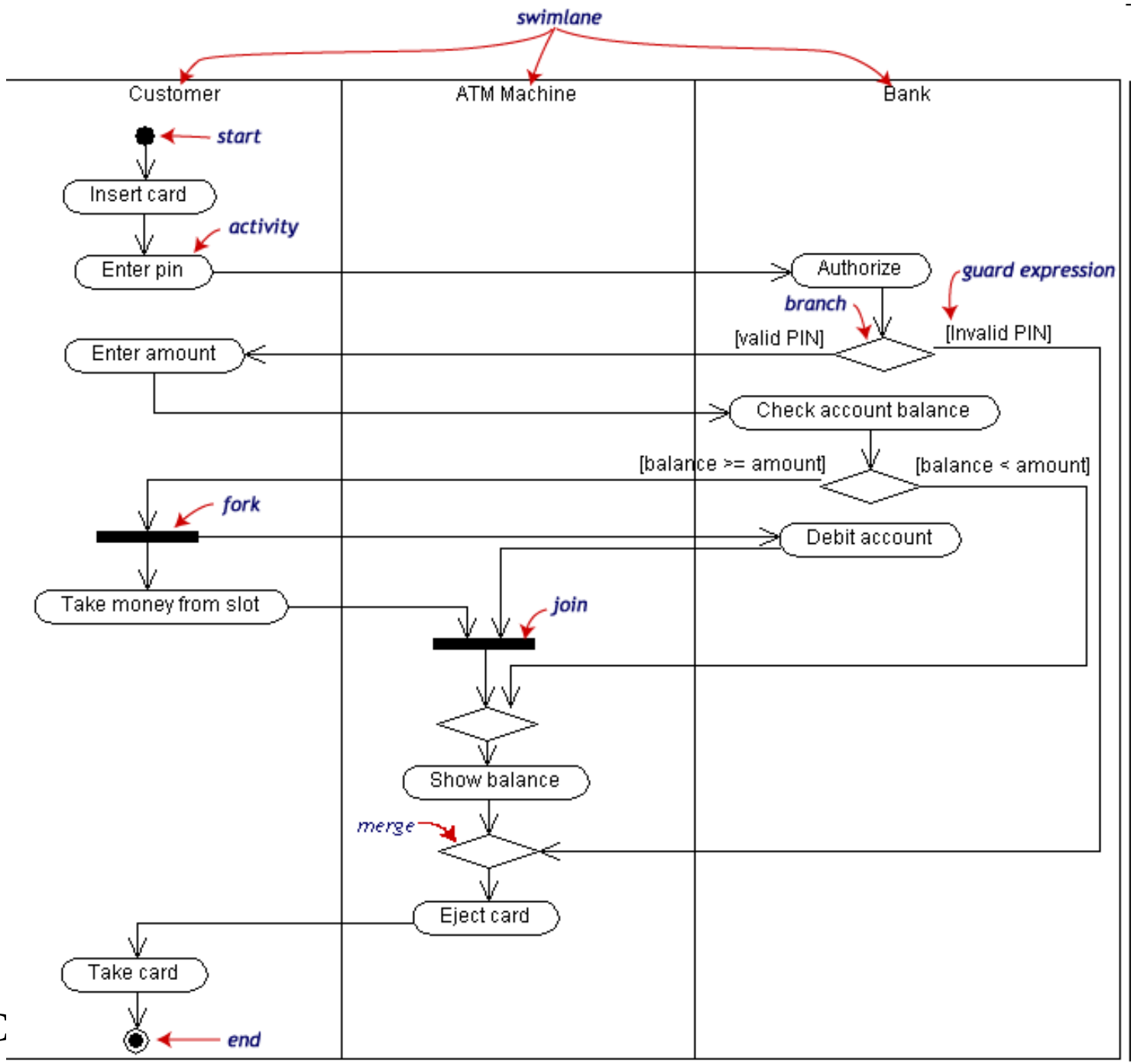
STATE DIAGRAMS



Lifetime of a login class.

- Show the state of a particular object throughout the system lifetime.
- Not necessary for all objects – just the critical ones.
- Super states – can be used to nest states to make diagram easier to read.

ACTIVITY DIAGRAMS



C

UML VERSIONS

- UML 1.0 was created through a collaboration of various organizations: Microsoft, IBM, HP, etc.
- Today, UML is on Version 2.0.
- It is managed by the Object Management Group – not for profit consortium. UML specification is available for download at http://www.uml.org/
- Tools that conform the latest 2.0 specification – Rational Software Architect, Sparx System Architect, StarUML, etc.

WHAT IS AN OBJECT?

- An entity that encapsulates data and behavior
- - Objects are categorized into *classes*
- - Each individual object is an *instance* of a
- class

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.**

WHAT IS ENCAPSULATION?

- The characteristic of object-orientation in which data and behavior are bundled into a class and hidden from the outside world

- Access to the data and behavior is provided and controlled through an object's *interface*

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

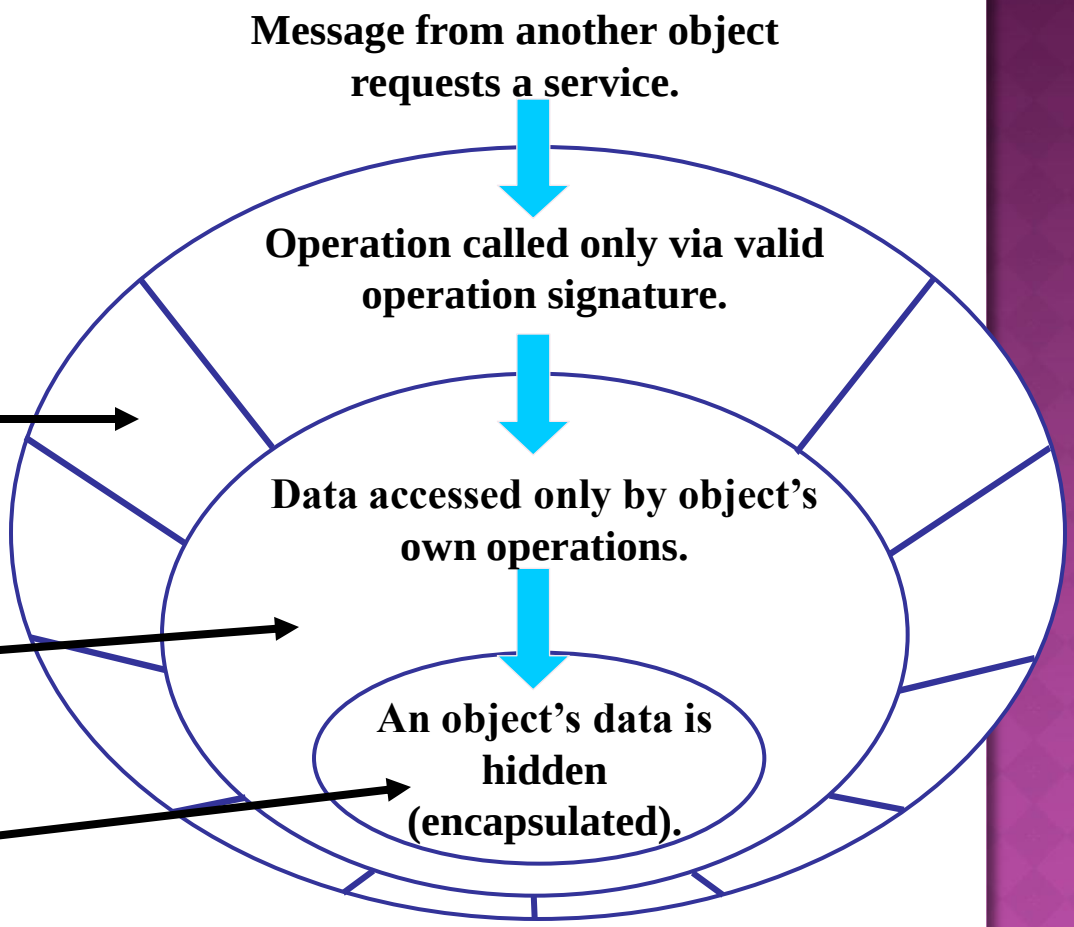
MESSAGE-PASSING AND ENCAPSULATION

‘Layers of an onion’ model of an object:

An outer layer of operation signatures…

…gives access to middle layer of operations…

…which can access inner core of data



WHAT IS A CLASS?

- A category of objects that share the same attributes, operations, relationships, and semantics
- All objects are instances of classes

CLASS AND INSTANCE

- All objects are *instances* of some *class*
- Class:

  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 (2004)

CLASS AND INSTANCE

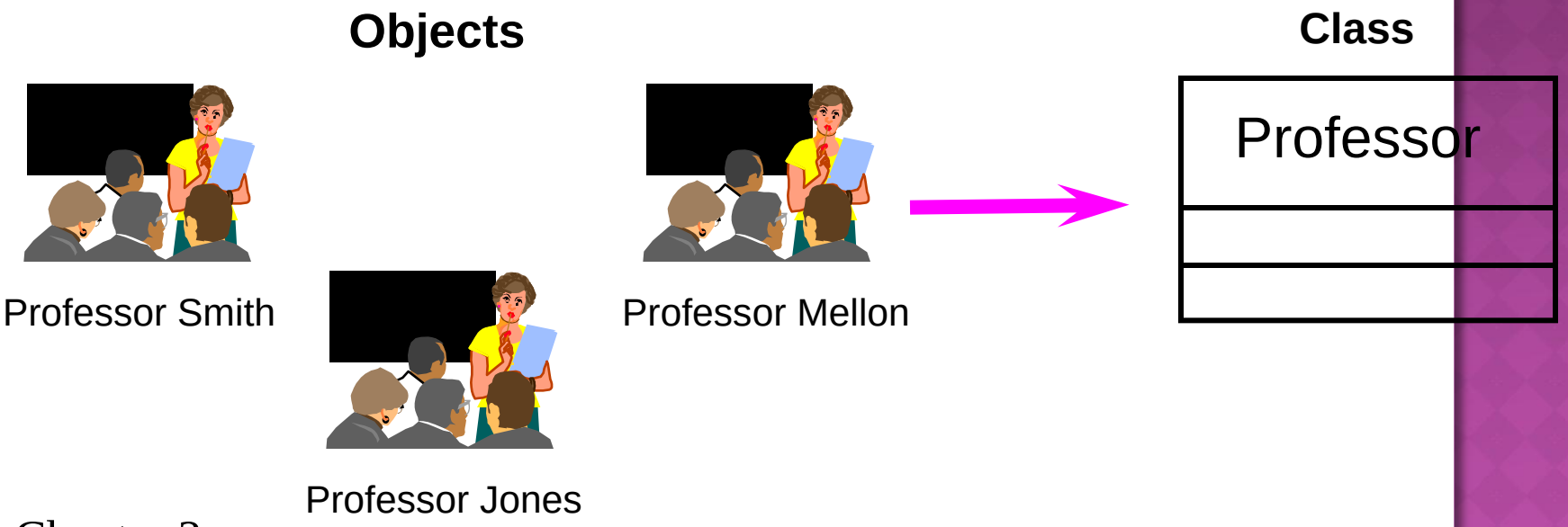
- An object is an instance of some class
- So, instance = object
  - but also carries connotations of the class to which the object belongs
- Instances of a class are similar in their:
  - *Structure*: what it *knows*, what information it holds, what links it has to other objects
  - *Behaviour*: what an object *can do*

Figure 2.4

An Illustration of a Class

THE RELATIONSHIP BETWEEN CLASSES AND OBJECTS

- A class is an abstract definition of an object
  - It defines the structure and behavior of each object in the class
  - It serves as a template for creating objects
- Objects are grouped into classes

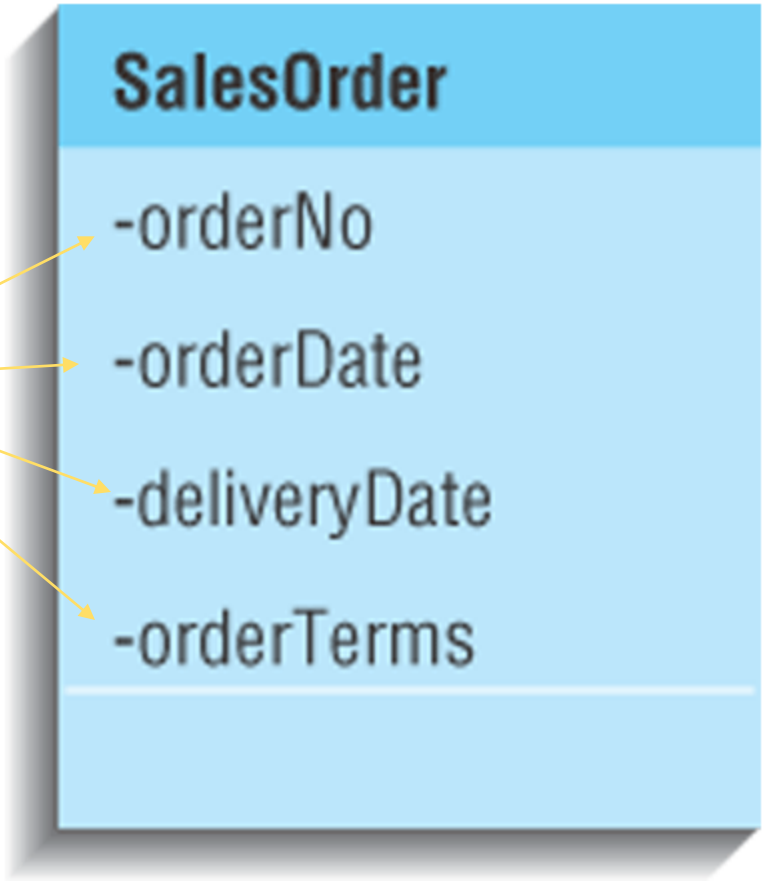


WHAT IS AN ATTRIBUTE?

- Attribute- a named property of a class that describes a range of values that instances of the attribute might hold

- Attributes are the way classes encapsulate data
- An attribute is a data value or state that describes an object and helps you to tell one object from another of the same class.

Figure 2.5
A Class with Attributes



Minus sign indicates these are private (hidden)

Attributes are properties containing values

WHAT IS AN OPERATION?

- A behavior of an object
- Implemented in classes are *methods*
- Methods are identified and invoked by their *signatures*, including name, parameters, and return type

Figure 2.6 An Operation with Signature and Method Written in PL/SQL

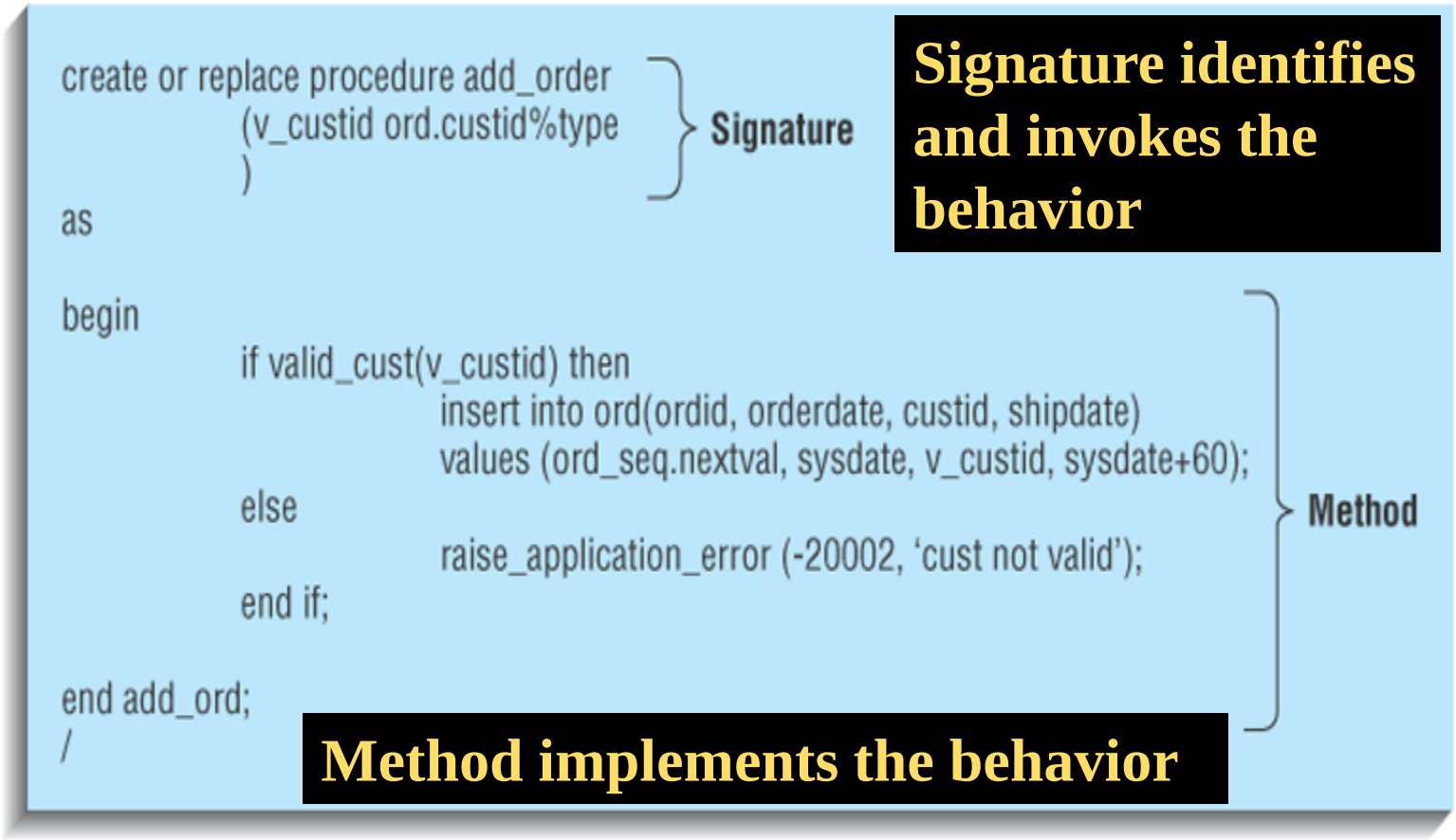
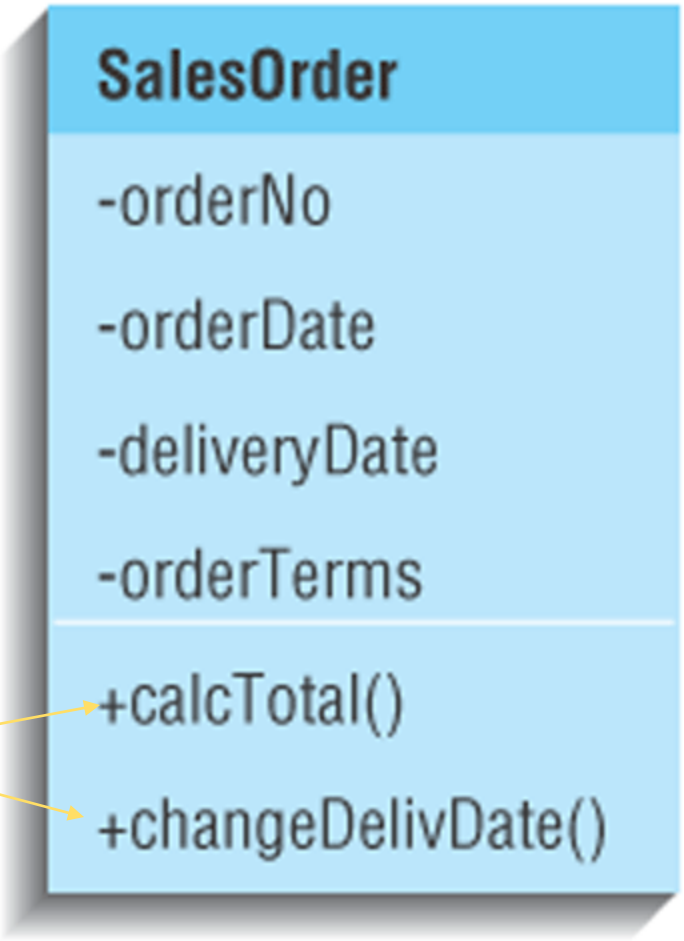


Figure 2.7
A Class with Attributes and Operations



Plus sign indicates these are public (accessible)

Method signatures

WHAT IS GENERALIZATION?

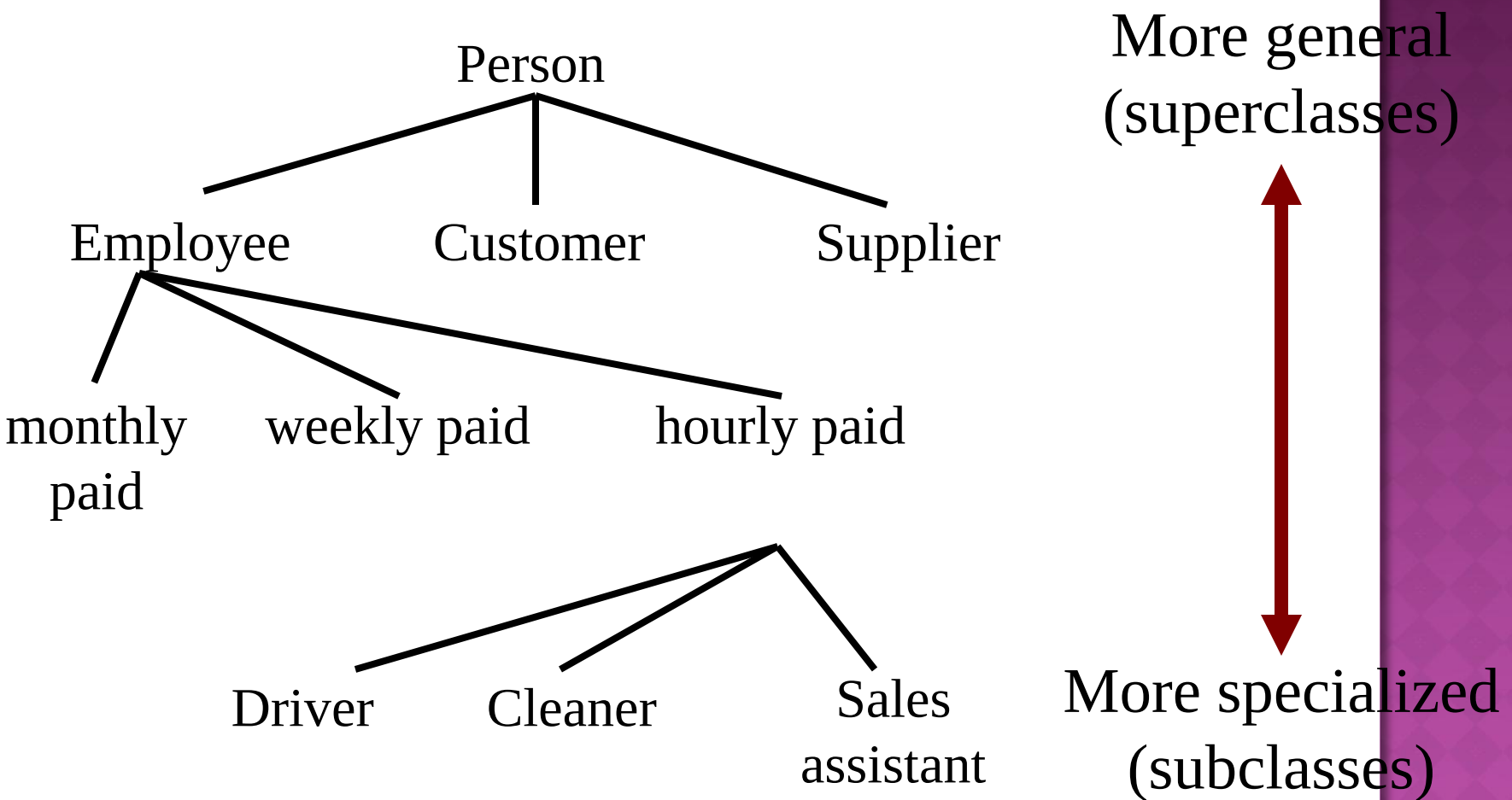
- A relationship between a more general (or parent) class and a more specific (or child) class

- The more specific class has additional attributes and operations

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



GENERALIZATION AND SPECIALIZATION

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

General (superclass)

Person
name date of birth gender title

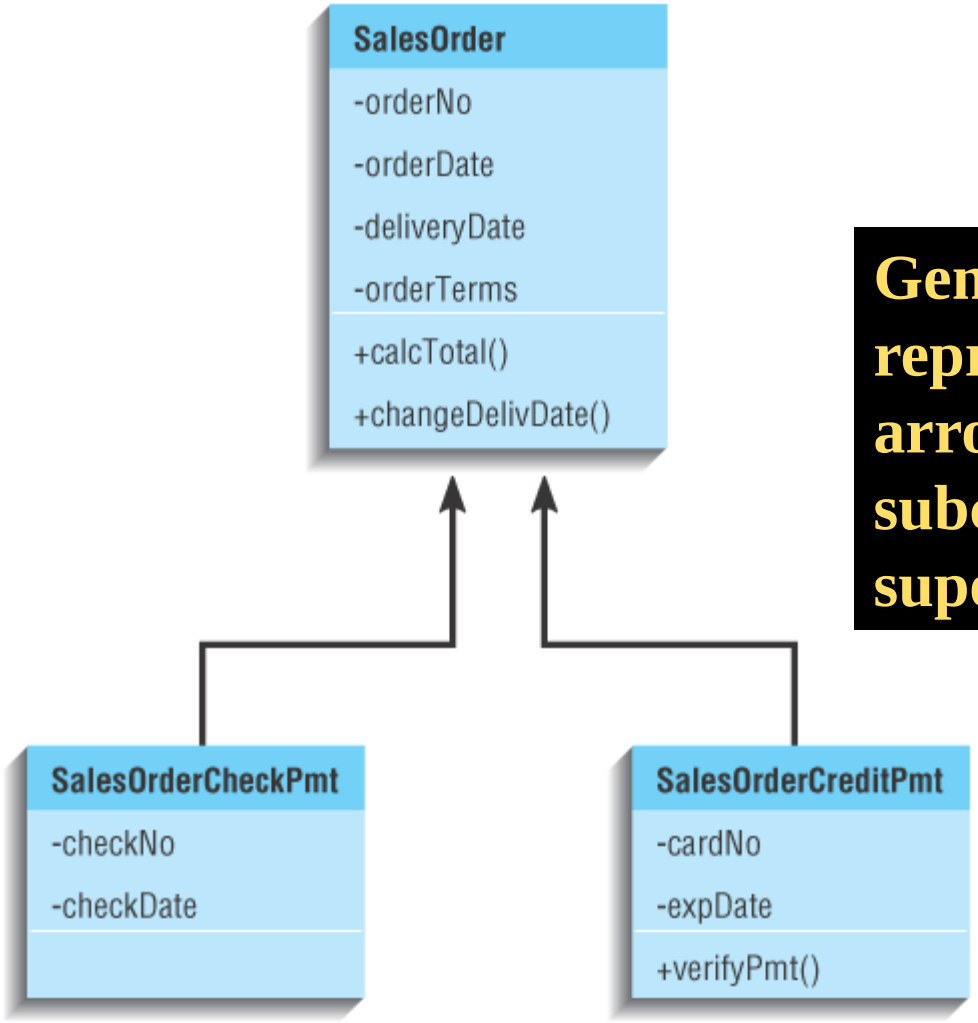
Specialized (subclass)

HourlyPaidDriver
startDate standardRate overtimeRate licenceType

WHAT IS INHERITANCE?

- The mechanism by which the more specific class in a generalization relationship includes the attributes and operations of the more general class

Figure 2.8 An Example of Generalization



Generalization represented by arrows from subclass to superclass

Subclasses inherit all attributes and operations of superclasses

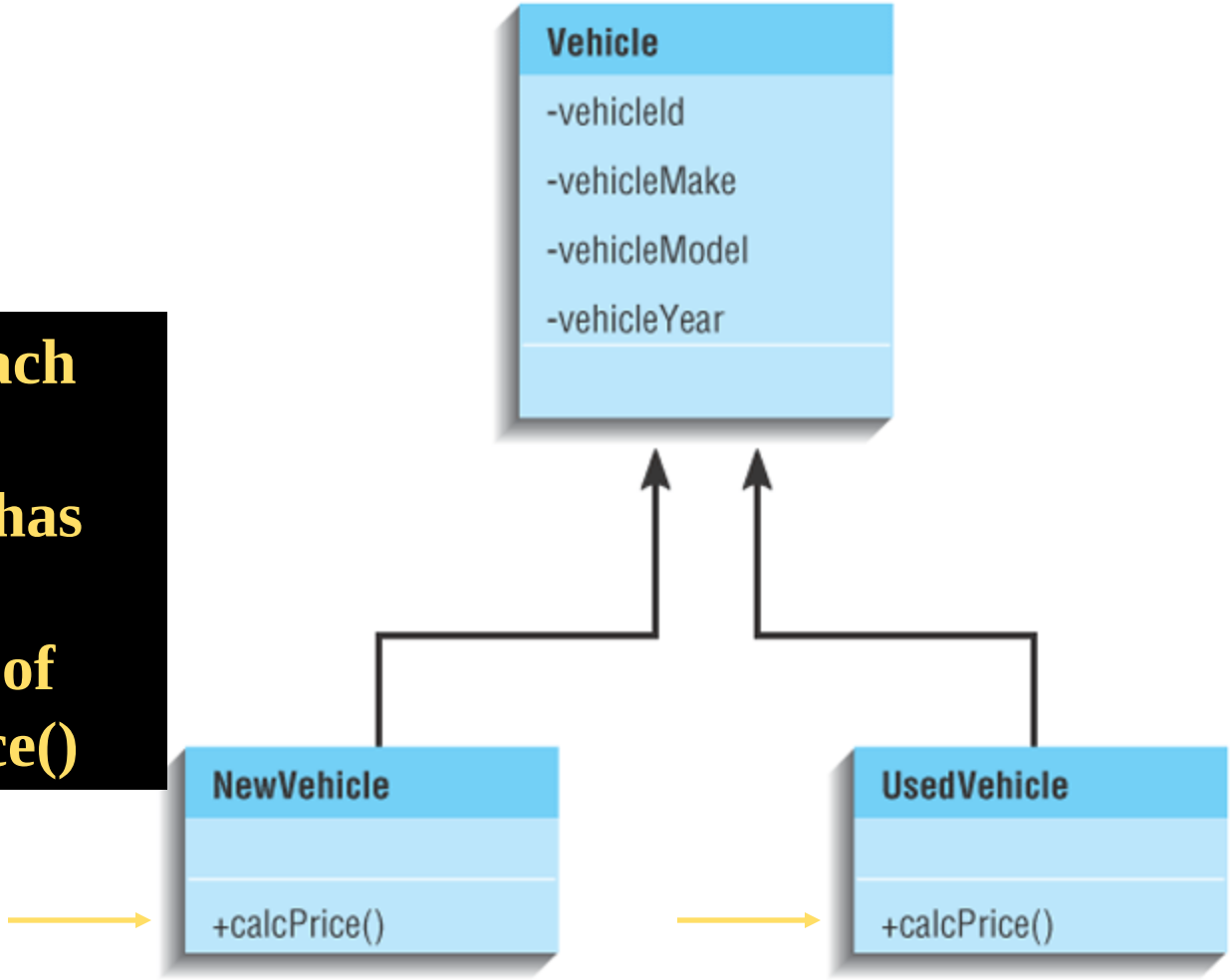
WHAT IS POLYMORPHISM?

- The ability for different classes of objects to respond to identical messages in different ways
- Polymorphism = “having many forms”
- Different behaviors for the same message

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

Figure 2.9 An Example of Polymorphism

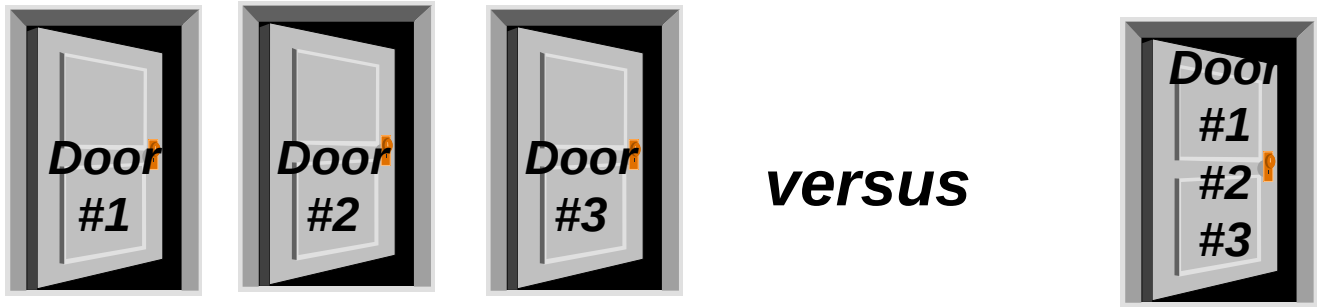


Here, each type of vehicle has its own version of calcPrice()

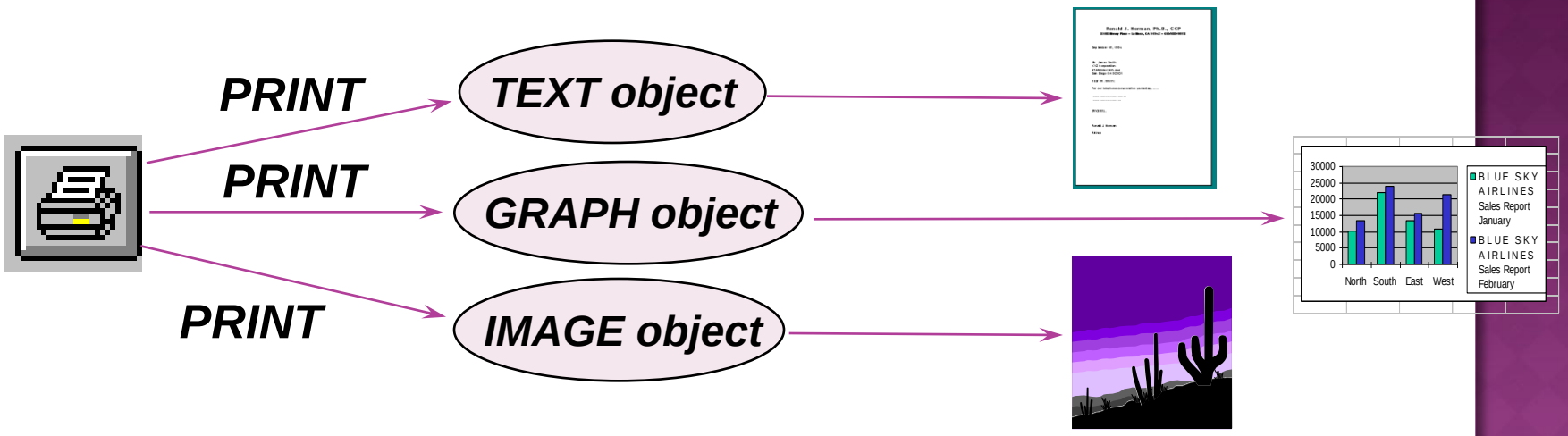
- **Polymorphism** ***("many forms")***

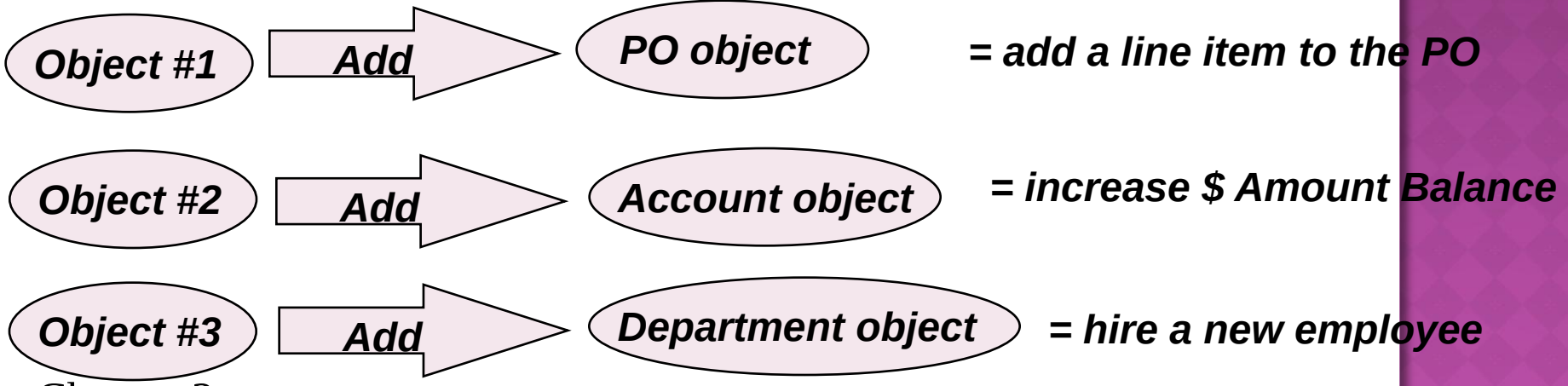
  - ***The ability to hide different implementations behind a common interface.***
  - ***The ability for two or more objects to respond to the same request, each in its own way.***

- *H O = water, ice, steam (liquid, solid, vapor)*
  2
- *Eating*



• Polymorphism 4 Two examples





WHAT IS A COMPONENT?

- A replaceable part of a system providing a clearly defined function through a set of interfaces
- Group of classes working together toward a common end; a subsystem

WHAT IS A COMPONENT?

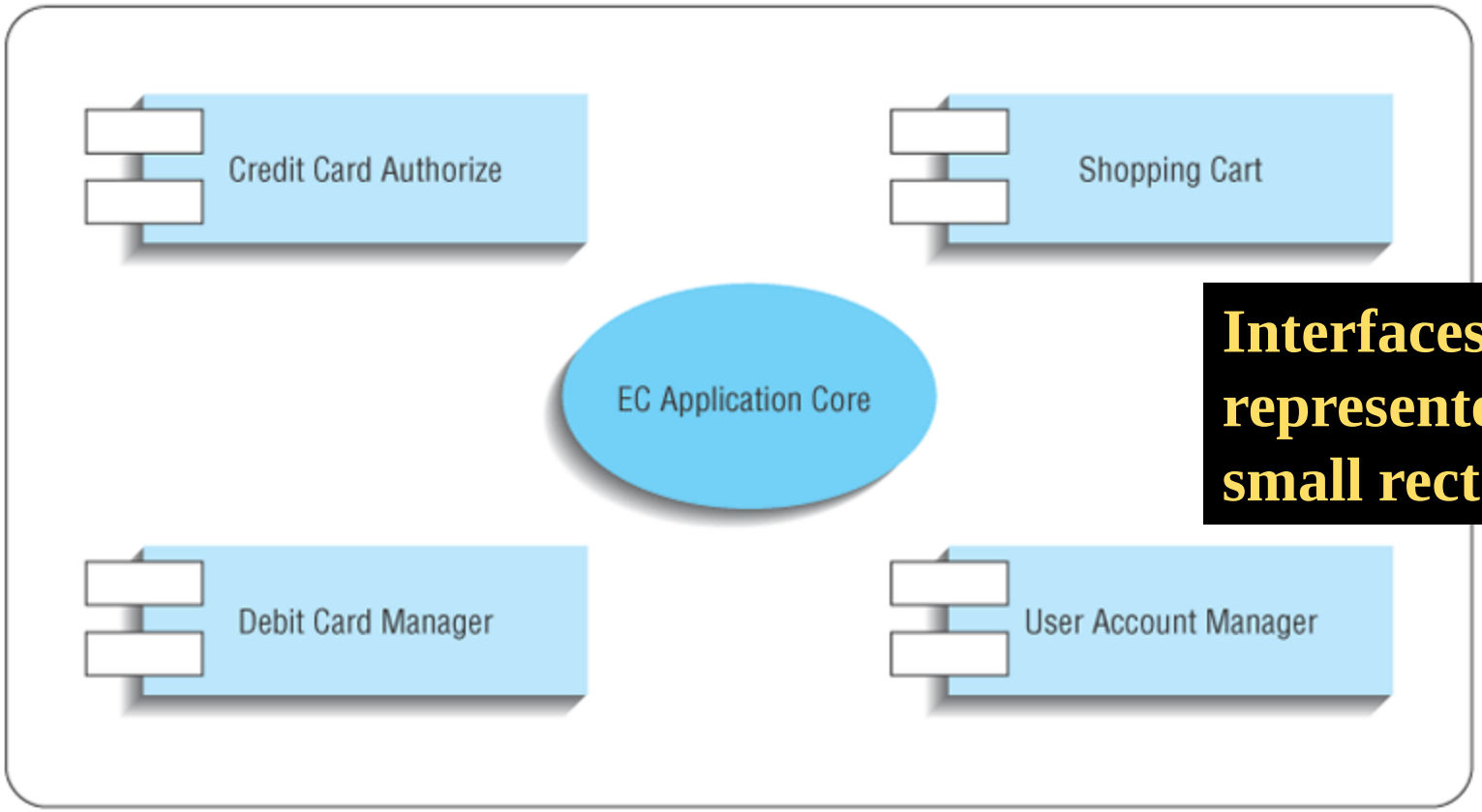
- A component is a collection of related classes that together provide a larger set of services.
- Components in your system might include applications, libraries, ActiveX controls, JavaBeans, daemons, and services.

WHAT IS AN INTERFACE?

- The mechanism by which users of a component invoke its behaviors and manipulate its properties

- The interface is implemented by method signatures
- An interface is a definition of a set of services provided by a component or by a class.

Figure 2.10 An EC System with Four Components

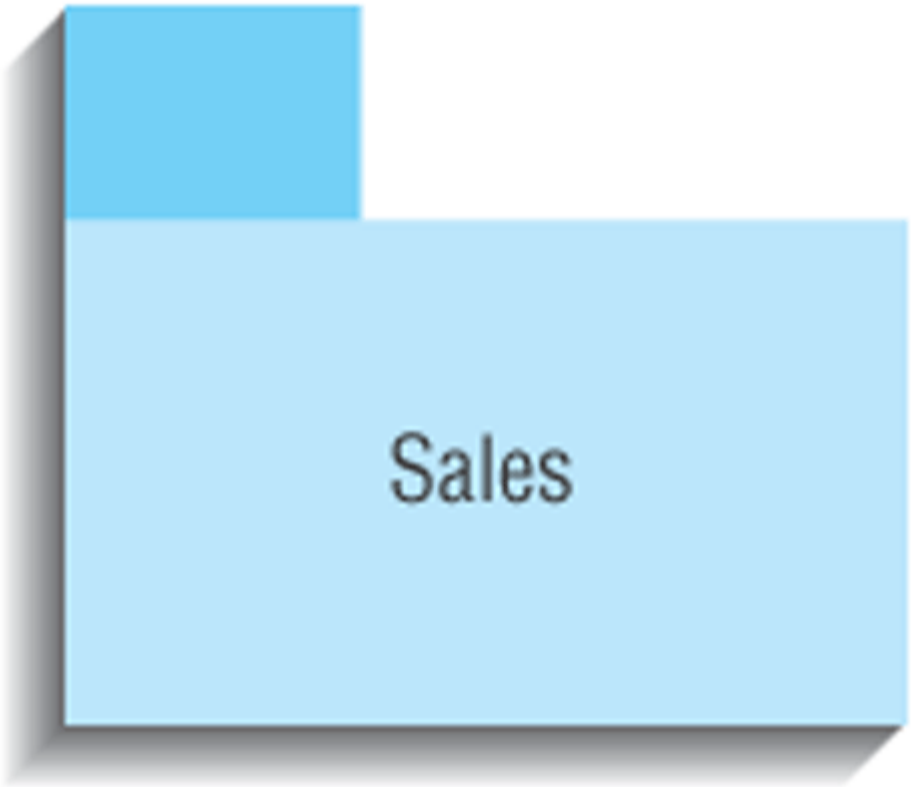


Interfaces are represented as small rectangles

WHAT IS A PACKAGE?

- A general-purpose mechanism for organizing elements into groups
- Group of classes sharing similar characteristics or purposes

Figure 2.11
An Example of a Package



WHAT IS AN ASSOCIATION?

- A relationship or link between instances of one, two, or more classes

- Three types:
  - Simple associations: no ownership
  - ***Aggregations***: part-whole relationships where the part can exist independently of the whole
  - ***Compositions***: part-whole relationships where the part and the whole are fully dependent on each other

Figure 2.12 An Example of an Association Between Two Objects

This is a binary association, showing roles and multiplicities

RECAP

- After studying this chapter we learned to:
  - Define an object.
  - Understand the terms *class*, *attribute*, and *operations*.
  - Explain generalization, polymorphism, and inheritance.
  - Define association.
  - Describe modeling and the Unified Modeling Language.