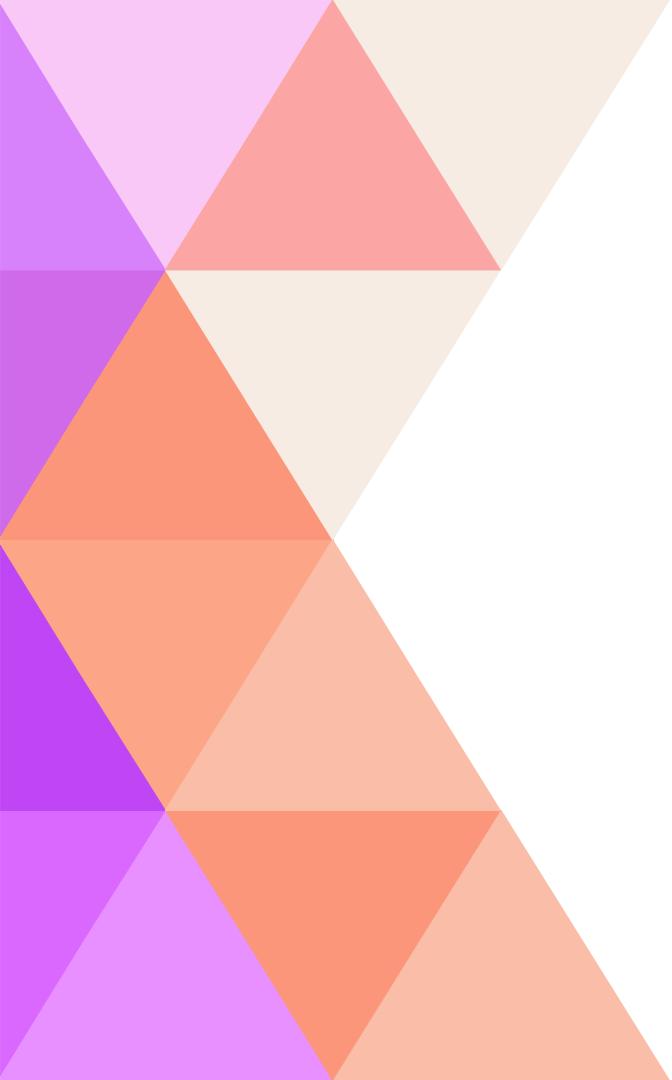


# **OBJECT ORIENTED ANALYSIS AND DESIGN**

Lecture 01





# Today's Outline

Administrative Stuff

Overview of 309

Introduction to Object Oriented Analysis  
& Design

Difference between SAD and OOAD

The background of the slide features a geometric pattern of overlapping triangles. There are several large triangles in the foreground, colored in shades of purple, pink, orange, and beige. Behind them are smaller triangles of the same colors, creating a layered effect.

Administrative Stuff

## Office hours

Office 6 (CS building)

Office hours: 3:00 to 4:00 pm

(Monday, Thursday, Friday)

# About the course

- Object Oriented Analysis & Design is an innovative way of thinking about problems using models organized in real world concepts.
- To study the fundamental construct of OOAD which combines both data structure and behavior in a single entity.
- To explain how a software design may be represented as a set of interacting objects that manage their own state and operations
- Various cases studies will be used throughout the course to demonstrate the concepts learnt.
- A strong in class participation from the students will be encouraged and required during the discussion on these case studies.

# Pre-requisites /Knowledge assumed

**Programming language** concepts

**Data structure** concepts

We assume you have the skills to code in any programming language therefore you

can **design, implement, test, debug, read, understand and document the programs.**

# Tentative Grading Policy

Assignments/Class Participation: 10 %

Quizzes: 10%

Mid Exam 1: 15 %

Mid Exam 2: 15 %

Final: 40%

Presentations + Projects: 10%

## An advanced level project

Should provides interesting realistic problem,

## Small 3-4 sized team

## Emphasis on good **OOAD** methodology

Homework's and deliverables tied to the project

# Course Material

You will have Presentations of each topic and ***reference books*** in PDF format will be available on slate.

## ***Text Books***

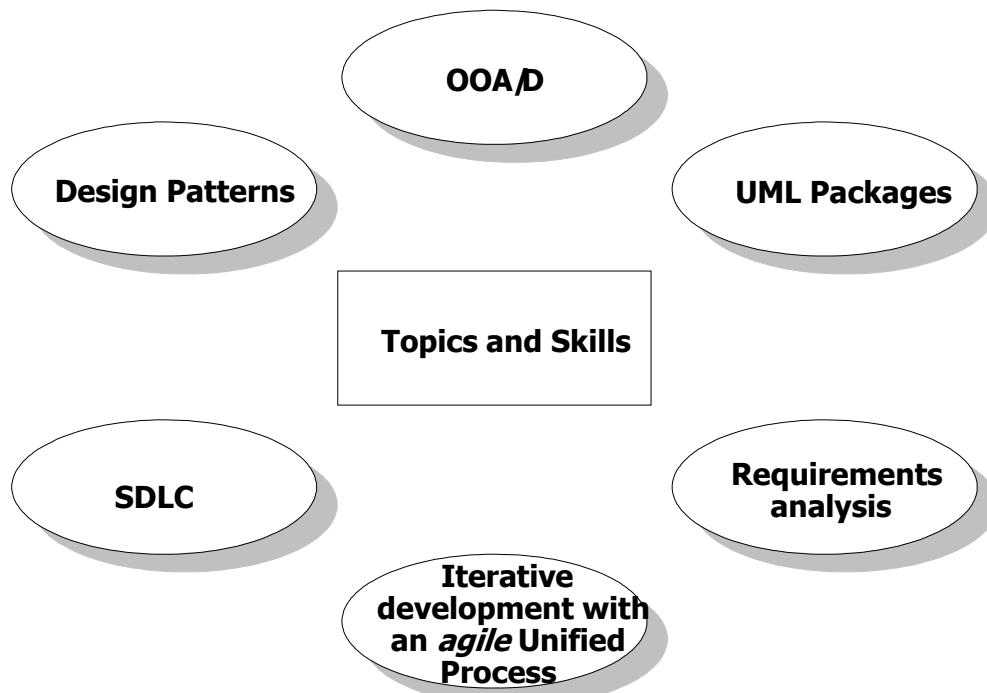
1. *UML 2 Toolkit* by Hans-Erik Eriksson, Magnus Penker, Brian Lyons, David Fado
2. *Applying UML and Patterns 3rd Edition* by Craig Larman

## ***Reference Books***

1. *UML and the Unified Process, Practical object-oriented analysis and design* by Jim Arlow, Ila Neustadt
2. *The Unified Modeling Language Reference Manual, 2nd edition* by James Rumbaugh, Ivar Jacobson and Grady Booch
3. *UML Distilled, 2nd Edition* by Martin Flower

Hands-on labs will be part of the course.

# Major Topics of the course



# Course Goals/Objectives

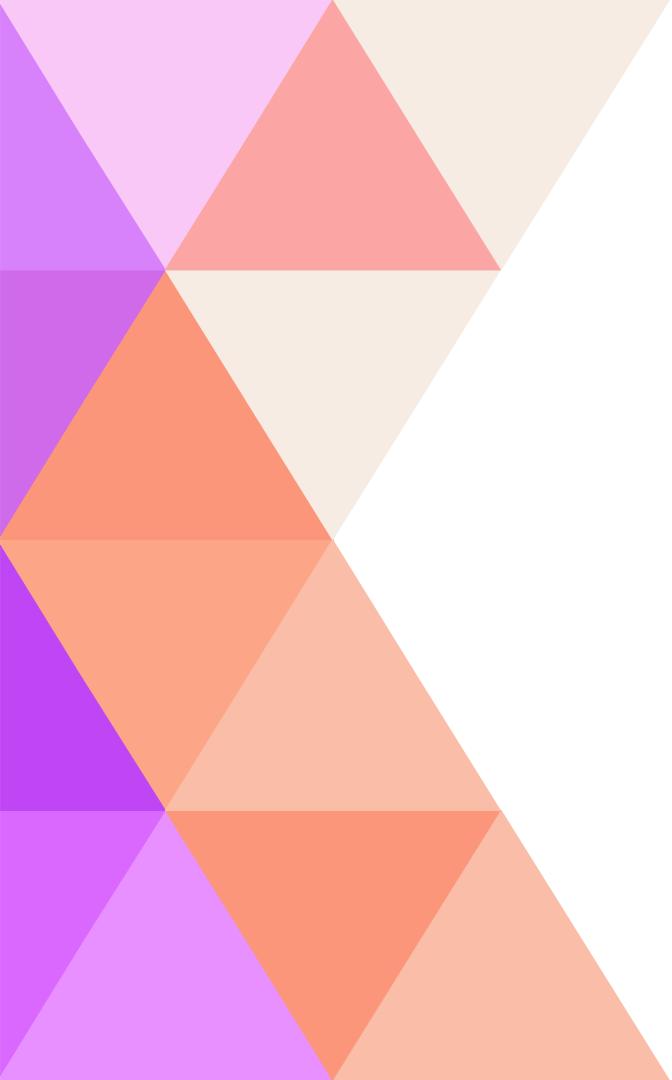
By the end of this course, you will be able to

Perform analysis on a given domain and come up with an Object Oriented Design (OOD).

Practice various techniques which are commonly used in analysis and design phases in the software industry.

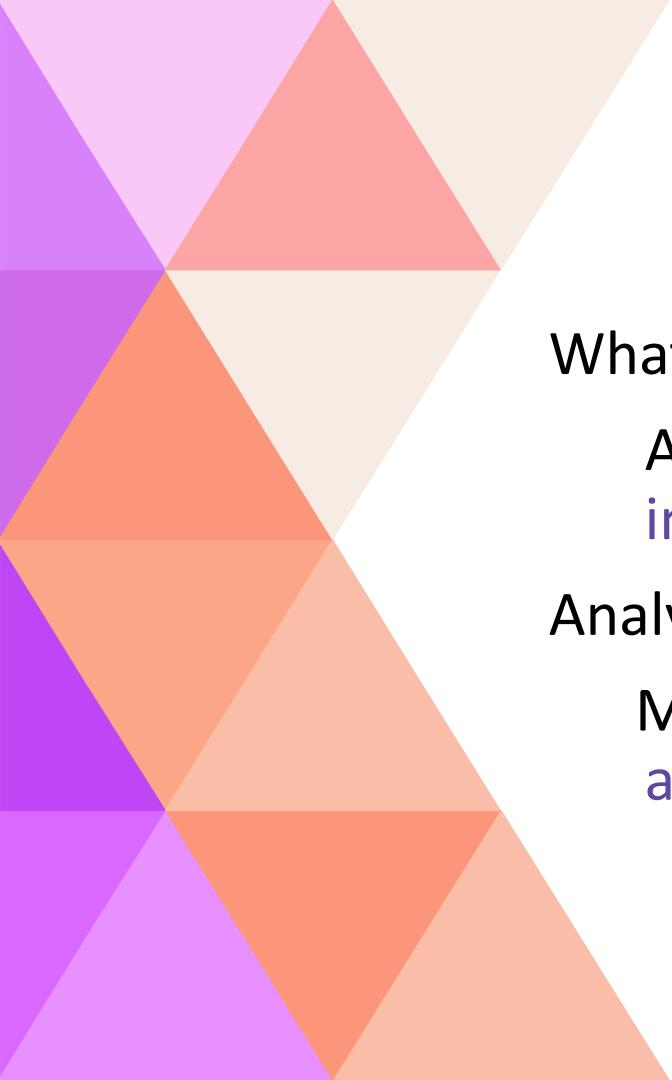
Use Unified Modeling Language (UML) as a tool to demonstrate the analysis and design ideas

Analyze, design and implement practical systems of up to average complexity within a team and develop a software engineering mindset



# Object-Oriented Analysis and Design (OOA/D)

- During **OOA**, the emphasis is on finding and describing the objects (or concepts) in the problem domain, i.e., **domain objects**.
- During **OOD** (or simply **object design**), the emphasis is on defining software objects and how they collaborate to fulfill the requirements.
- During **OOP** (OO Programming) or **Implementation**, design objects are implemented in a programming language.
  - Implementation is also known as **Coding** or **Construction**.



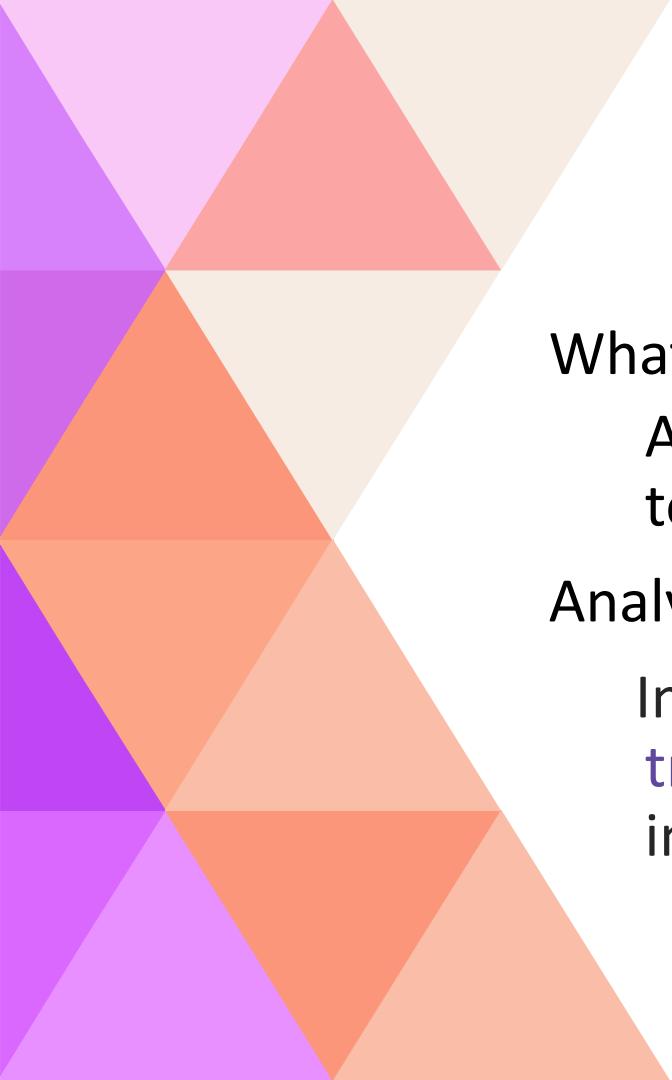
# What is System Analysis and Design?

What is a system?

All activities that go into producing an information system solutions

Analysis and Design of the system?

Mainly deals with the system development activities



# What is Software Analysis and Design?

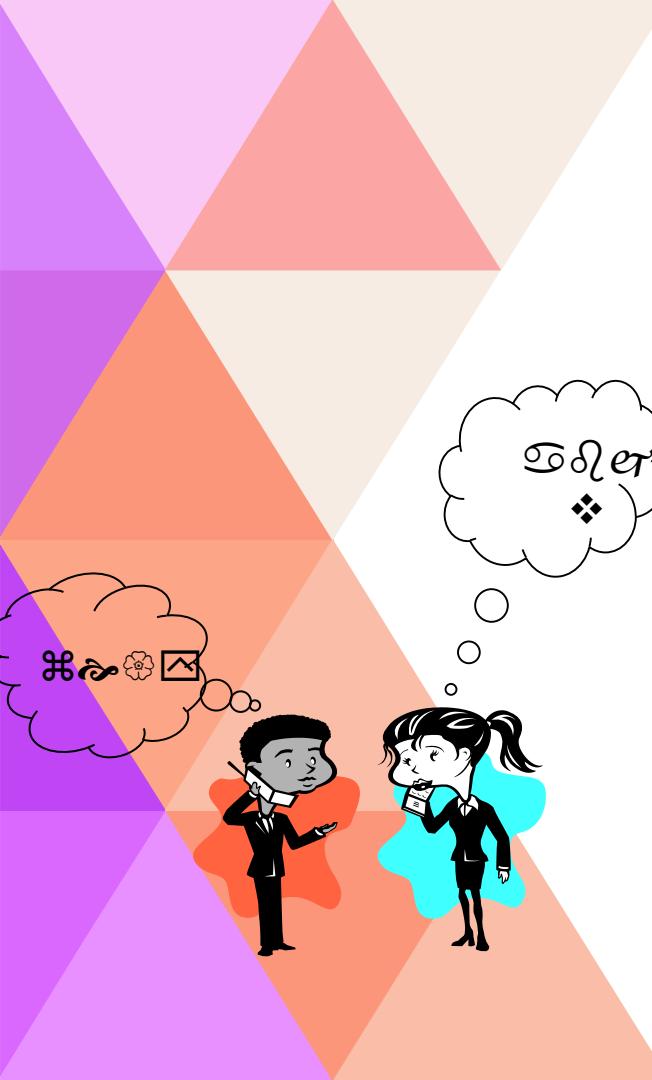
What is a software?

A series of processes that, if followed, can lead to the **development** of an **application**.

Analysis and Design of the software?

Includes all activities, which help the **transformation** of requirement specification into implementation.

# Why Object-Oriented?



*"The "software crises" came about when people realized the major problems in software development were ... caused by **communication** difficulties and the management of **complexity**" [Budd]*

*The Whorfian Hypothesis:*

*Human beings ... are very much at the mercy of the particular language which has become the medium of expression for their society ... the 'real world' is ... built upon the language habits ... We cut nature up, organize it into concepts, and ascribe significances as we do, largely because we are parties to an agreement to organize it in this way ... and is codified in the patterns of our language.*

***What kind of language can alleviate difficulties with communication & complexity hopefully well?***

# Why Object-Oriented?

– Consider Human Growth & Concept Formation

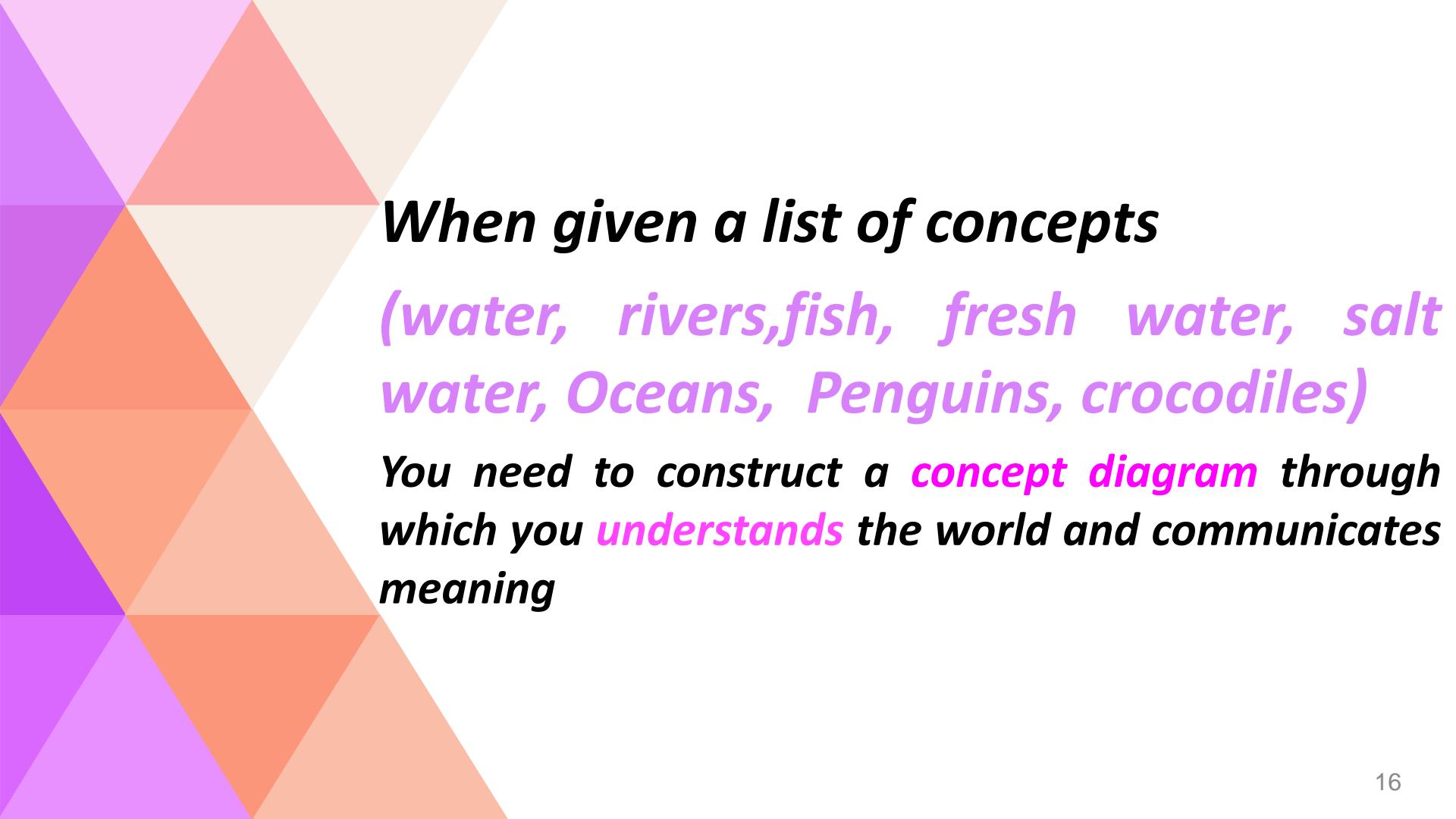
Communication & complexity about the problem and the solution,  
all expressed in terms of **concepts** in a language!

But then, What is CONCEPT? [Martin & Odell]

*Consider Human Growth & Concept Formation*

Stage	concepts
<i>infant</i>	<i>the world is a buzzing confusion</i>
<i>very young age</i>	<i>"blue" "sky" (individual concepts)</i> <i>"blue sky" (more complex concept)</i> <i>hypothesis: humans possess an innate capacity for perception</i>
<i>getting older</i>	<i>increased meaning, precision, subtlety, ...</i> <i>the sky is blue only on cloudless days</i> <i>the sky is not really blue</i> <i>it only looks blue from our planet Earth</i> <i>because of atmospheric effects</i>

*Concept formation: from chaos to order!*



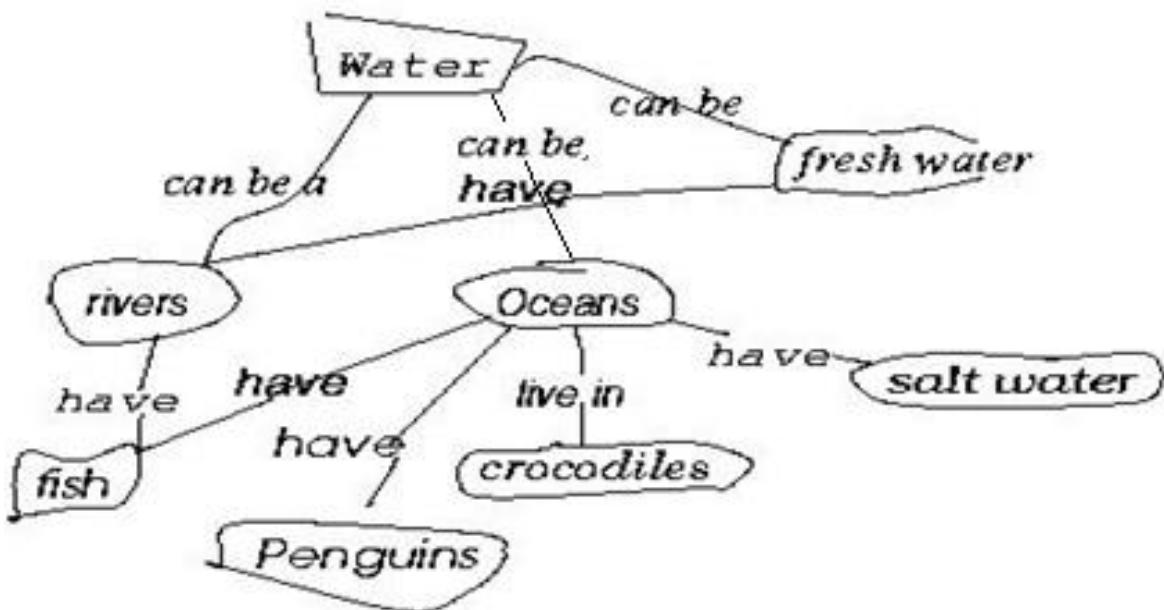
***When given a list of concepts  
(water, rivers,fish, fresh water, salt  
water, Oceans, Penguins, crocodiles)***

***You need to construct a concept diagram through  
which you understands the world and communicates  
meaning***

# Why Object-Oriented?

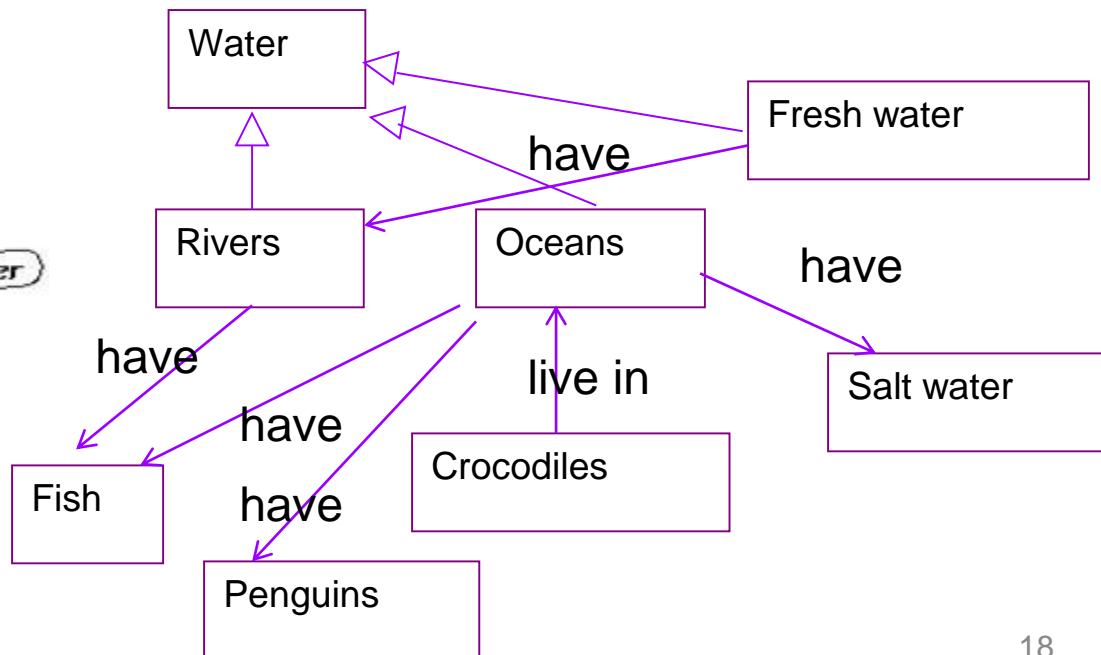
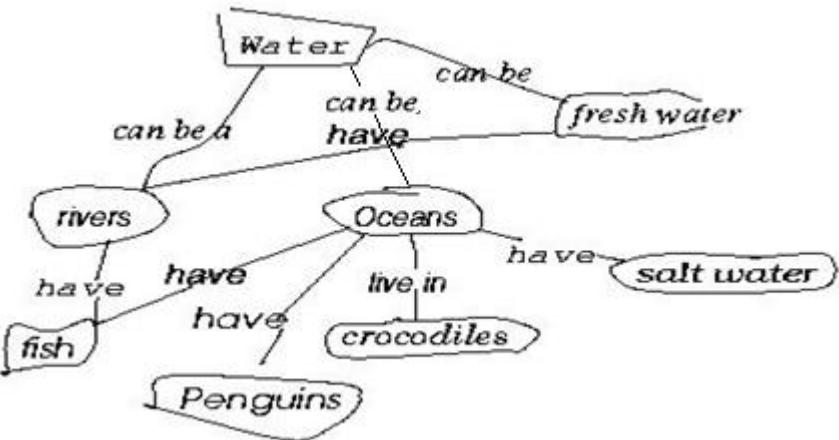
- concepts and objects

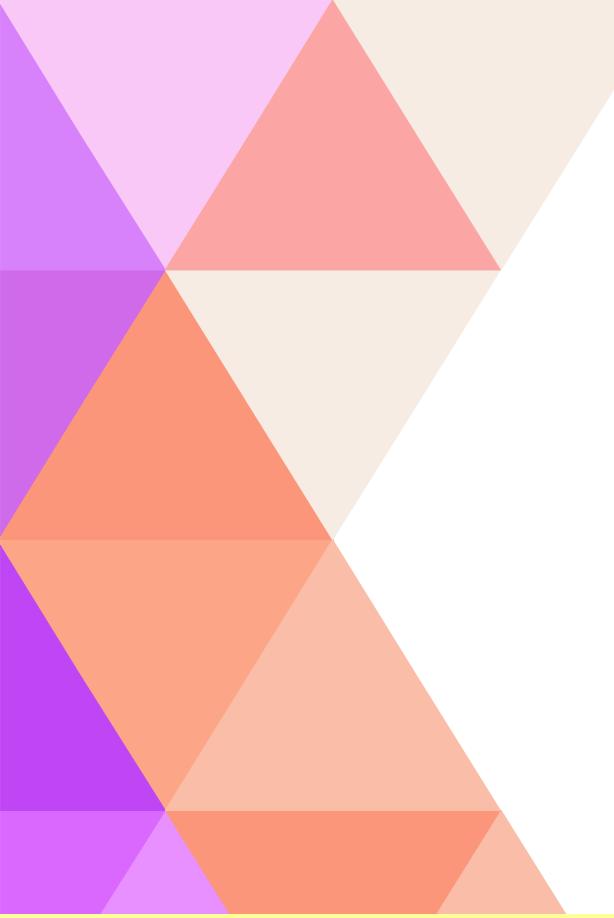
*So, concepts are needed to bring order ... into*



# Why Object-Oriented? ... for Conceptual ... Modeling Reasons

*What kind of language can be used to create this concept diagram, or your's mental image?*





# Why Object-Oriented?

What is a *model* and why?

- A model is a simplification of reality.  
*E.g., a miniature bridge for a real bridge to be built*
  - *Well...sort of...but not quite*
  - A model is *our* simplification of *our perception* of reality  
(that is, if it exists, otherwise it could be a mere illusion).  
communication is not about reality but about your/my/his/her perception of reality => validation and verification hard but needed
- A model is an *abstraction* of something for the purpose of *understanding*, be it the problem or a solution.

- To understand *why* a software system is needed, *what* it should do, and *how* it should do it.
- To communicate our understanding of why, what and how.
- To detect commonalities and differences in your perception, my perception, his perception and her perception of reality.
- To detect misunderstandings and miscommunications.

# **What is Object-Orientation?**

## **- What is Object?**

An object is any abstraction that models a single thing.

A representation of a specific entity in the real world.

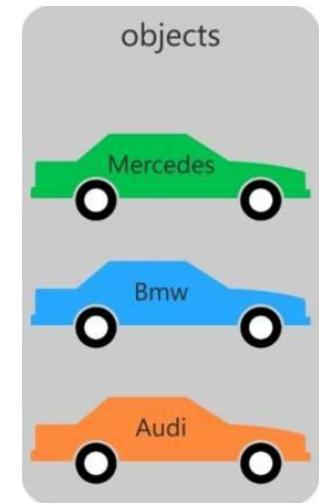
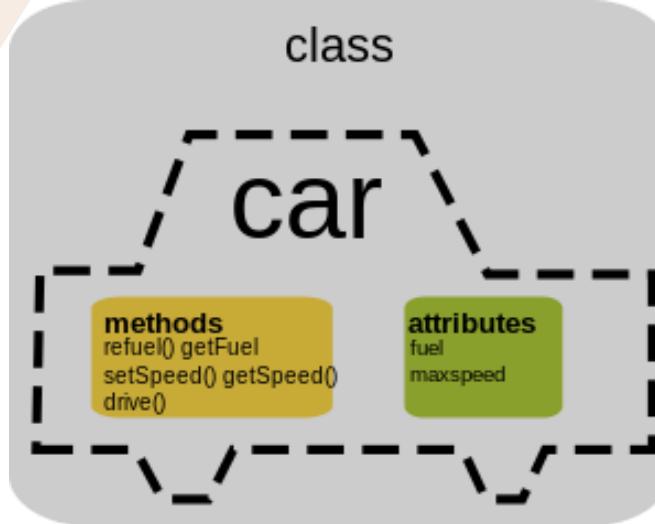
A structure that has identity and properties and behavior.

May be tangible (physical entity) or intangible.

Examples: specific citizen, agency, job, location, order, etc.

It is an instance of a collective concept, i.e., a class.

# What is Object-Oriented?



## What is *CLASS*?

a collection of objects that share common properties, attributes, behavior, in general.

A collection of objects with the same data structure (attributes, state variables) and behavior (function/code/operations) in the solution space.

## Classification

Grouping of common objects into a class

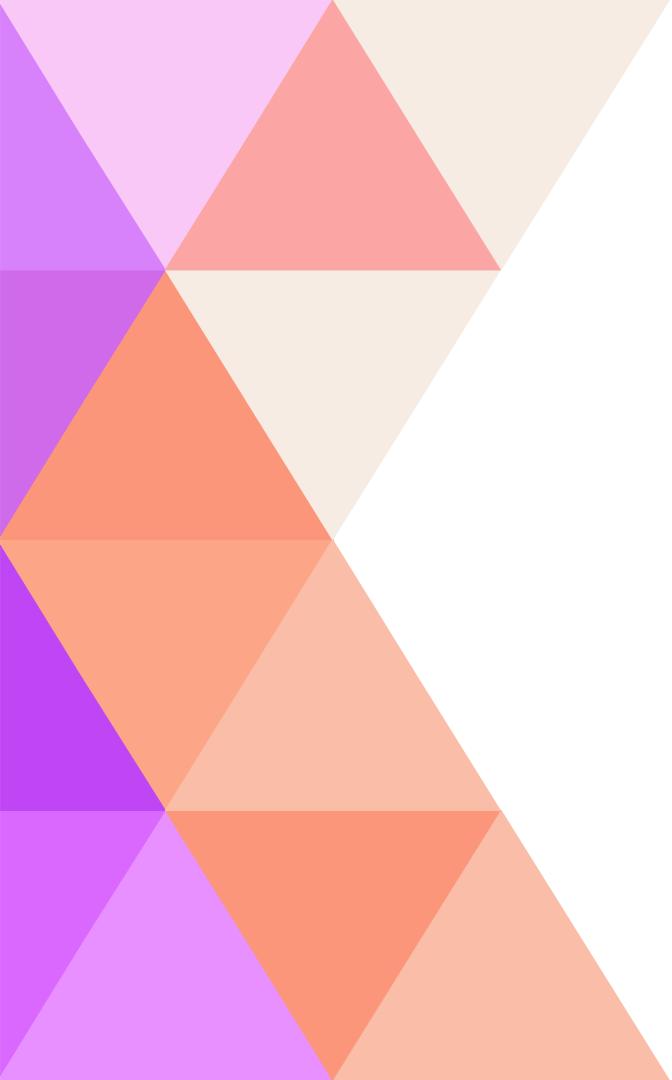
## Instantiation.

The act of creating an instance.



What are my  
Attributes and  
Operations??





# Two Orthogonal Views of the Software

- There are two different ways to view software construction:
    - We can focus primarily on the function  
*(Traditional System Development)*
    - We can focus primarily on the data  
*(Object Oriented System Development)*
- OR

# What is the Structure Analysis & Design (SAD)?

**Functional view  
of the problem**

Traditional systems development technique that is time tested and easy to understand

Transform data into information, called as **process centered technique**.

**Divide and Conquer**

**Top Down Approach**

Uses set of process models to describe a system graphically

Divide large, complex problem into smaller, more easily handled ones.

# What is the Structure Analysis & Design?

**Establish complete requirement documentation**

**Establish concrete requirement specification**

Structured analysis is a set of techniques and graphical tools that allow the analyst to develop a new kind of system specification that are easily understandable to the user. Analysts work primarily with their wits, pencil and paper.”

**Improve Quality and reduce risk**

**Focus on reliability, flexibility & maintainability**

# Structure Analysis & Design

## Advantages

- ⑩ visual, so it is easier for users/programmers to understand
- ⑩ Makes good use of graphical tools
- ⑩ A mature technique
- ⑩ Process-oriented approach is a natural way of thinking
- ⑩ Flexible
- ⑩ simple and easy to understand and implement

## Disadvantages

- ⑩ Not enough user-analyst interaction
- ⑩ It depends on dividing system to sub systems but it is to decide when to stop decomposing.

# What is Object Oriented Analysis & Design (OOAD)?

Object-Oriented analysis and design thoroughly represent complex relationships, as well as represent data and data processing with a consistent notation

**Essential for robust, well designed software**

**Emphasize on finding and describing objects**

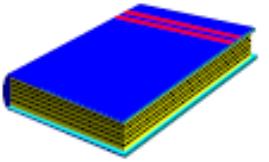
**Blend analysis and design in evolutionary process**

**Deals with complexity inherent in real world**

# Object Oriented Analysis and Design

## From Analysis to Implementation

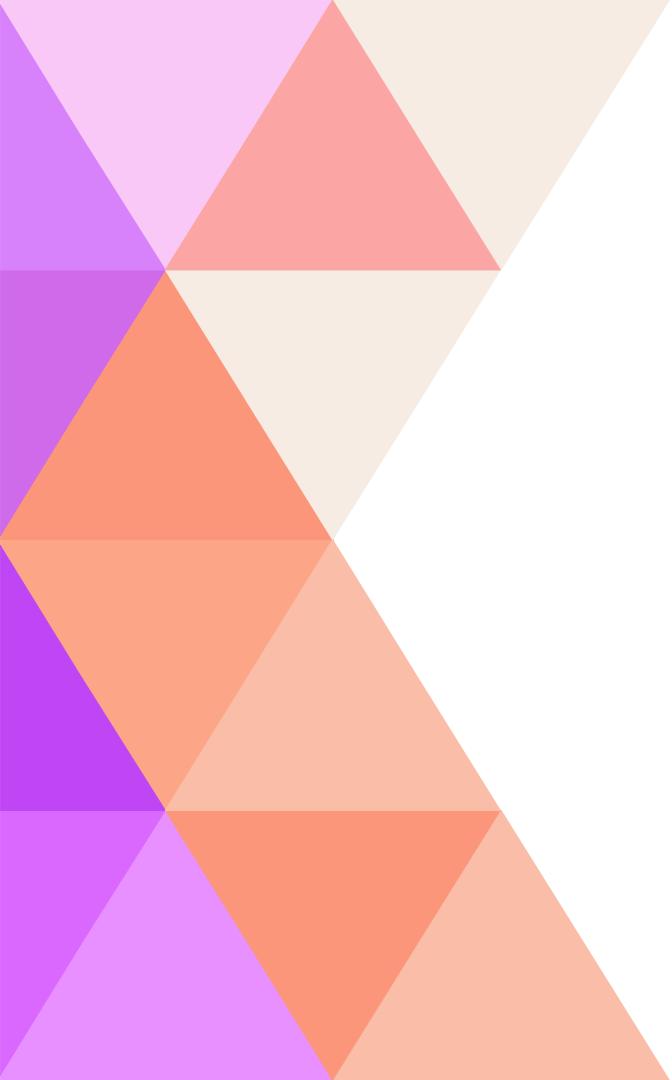


<p>Domain Concept Ex: Book (Concept)</p> 	<p>Logical Software Objects</p> <p>Book</p> <p>Attribute: Title</p> <p>Method: Display()</p>	<p>Representation in an OO Programming Language</p> <pre>Public Class Book {     Private String Title;     Public void Display(); }</pre>
--	--	---

# Similarities and difference between SAD and OOAD

## Key Differences Between Structured and Object-Oriented Analysis and Design

	Structured	Object-Oriented
Methodology	SDLC	Iterative/Incremental
Focus	Processs	Objects
Risk	High	Low
Reuse	Low	High
Maturity	Mature and widespread	Emerging (1997)
Suitable for	Well-defined projects with stable user requirements	Risky large projects with changing user requirements



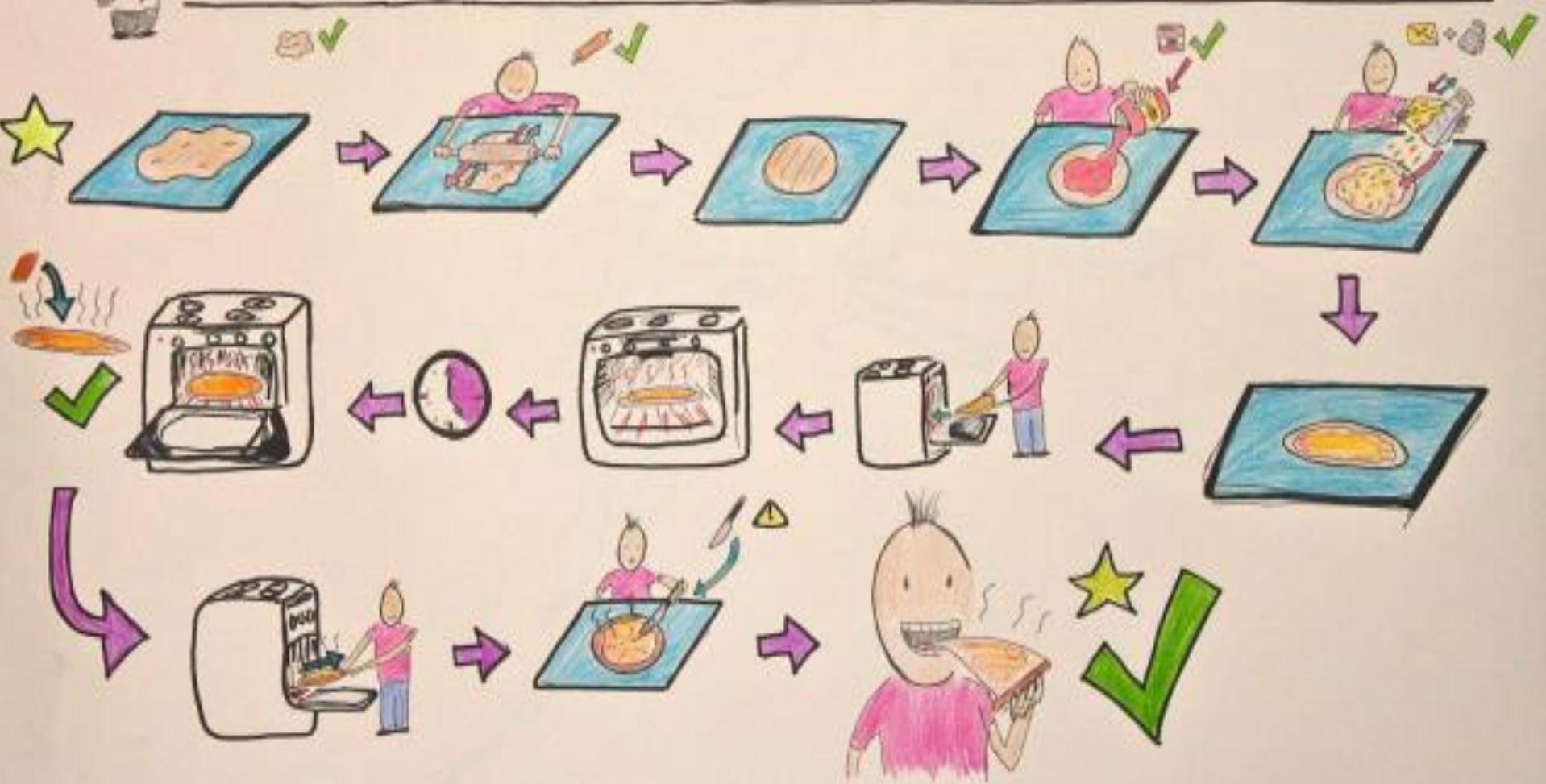
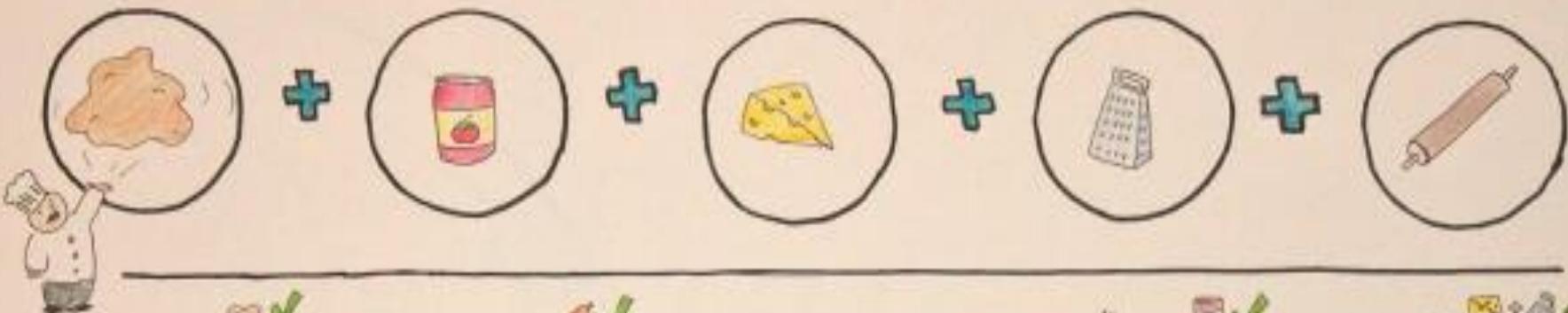
# Case Study: A PAYROLL PROGRAM

- Consider a payroll program that processes employee records at a small manufacturing firm. The company has several classes of employees with particular payroll requirements and rules for processing each. The company has three types of employees:
  - *Managers* receive a regular salary.
  - *Office Workers* receive an hourly wage and are eligible for overtime after 40 hours.
  - *Production Workers* are paid according to a piece rate.

You need to walk through traditional (SAD) and Object Oriented (OOAD) to highlight their differences.

# **SYSTEM DEVELOPMENT LIFECYCLE**

**Lecture 2**



# The System Development Life Cycle

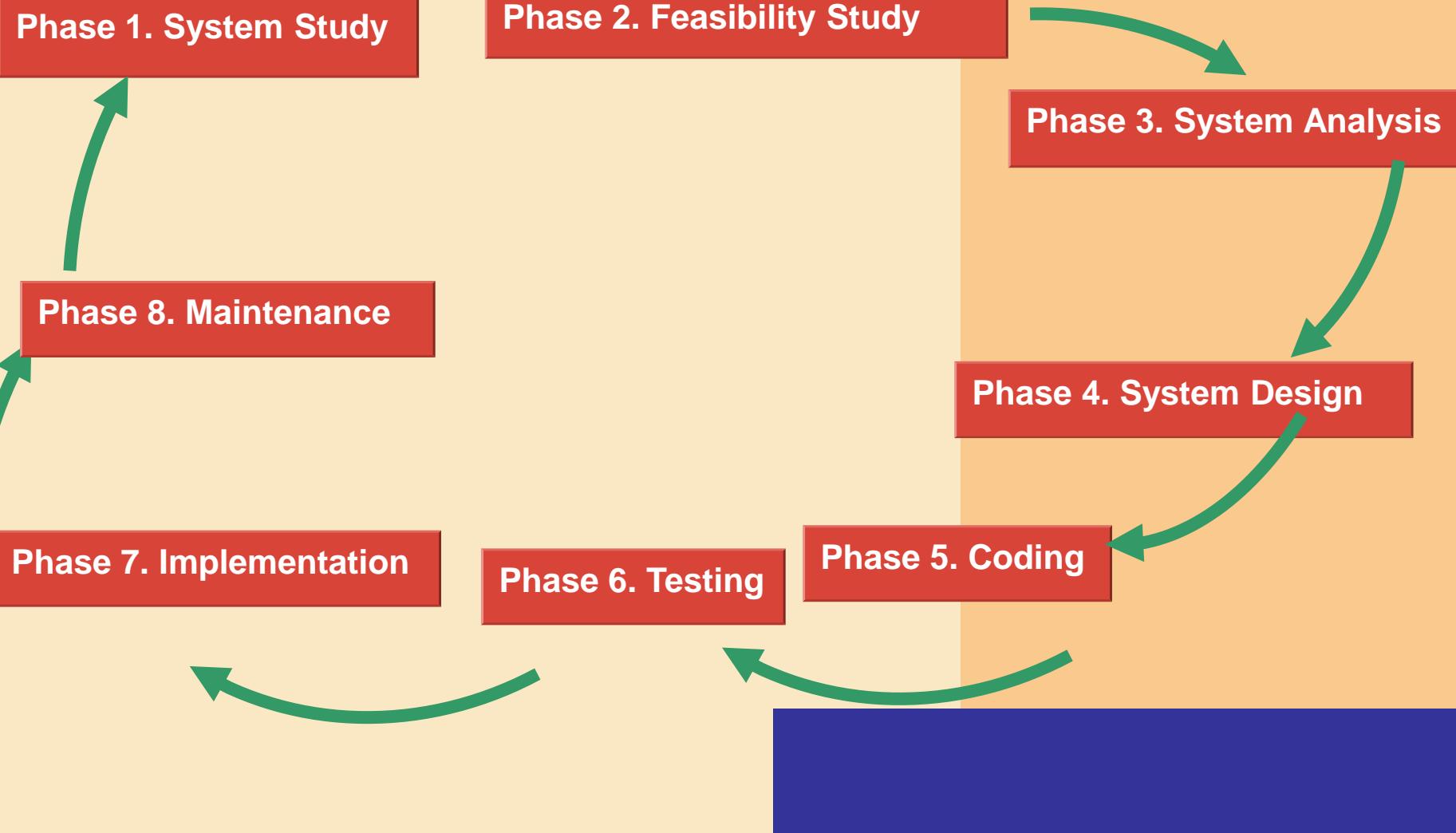
## What is an SDLC?

an organized process of developing and maintaining systems.

Combination of various activities

# The System Development Life Cycle

What are the phases of the system development cycle?



# The System Development Life Cycle

## What are guidelines for system development?

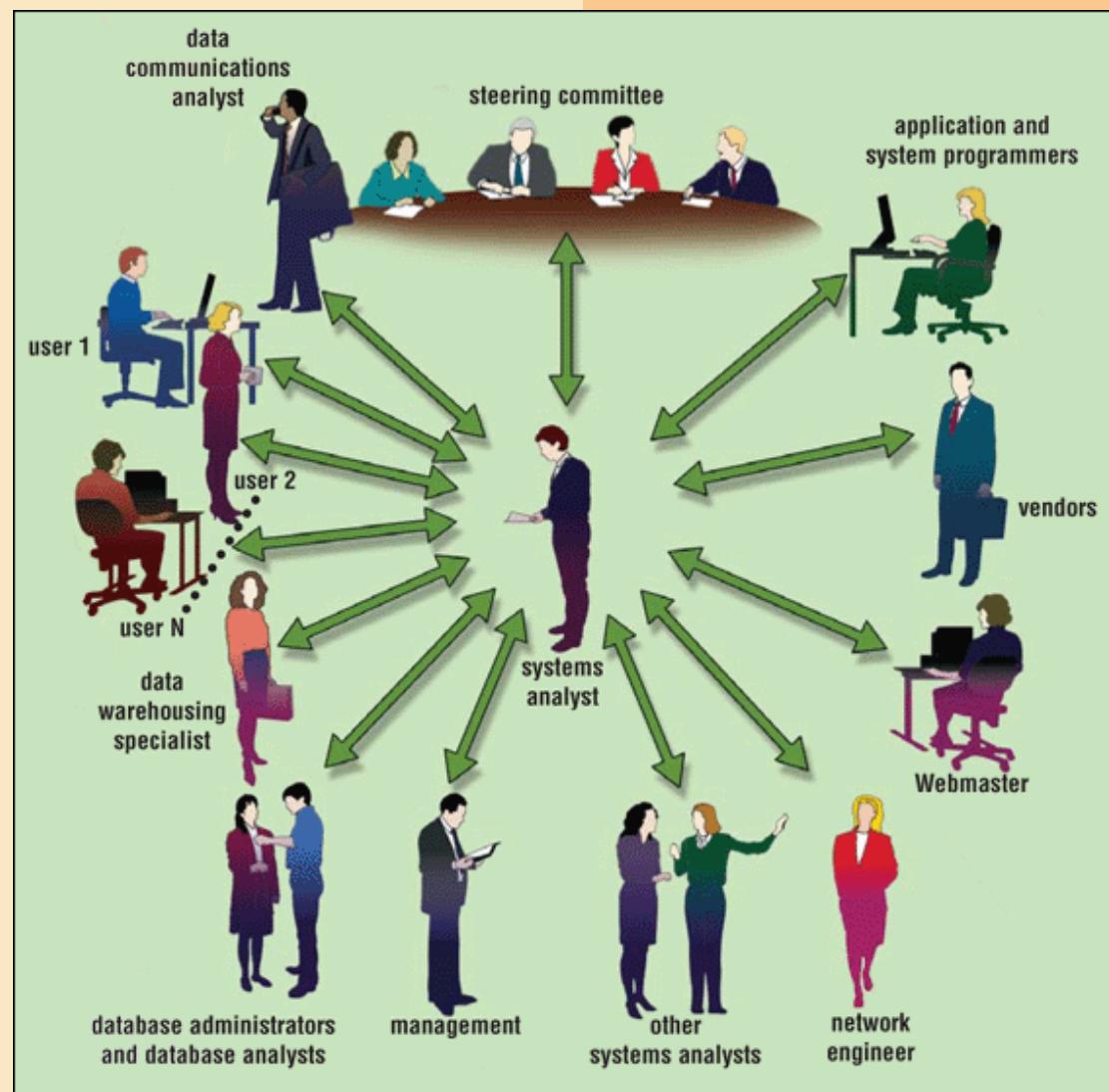
Arrange tasks into **phases**  
(groups of activities)

Involve **users** (anyone for whom  
system is being built)

Develop clearly defined **standards** (procedures  
company expects employees to follow)

# The System Development Life Cycle

Who participates in the system development life cycle?



# The System Development Life Cycle

## What is the project team?

Formed to work on project from beginning to end

Consists of users, systems analyst, and other IT professionals

Project leader—one member of the team who manages and controls project budget and schedule

# The System Development Life Cycle

## What is a System Study?

**Clear Picture of what  
actually the physical  
system is**

**Two Phases**

**Scope  
Identification**

**Requirement  
Identification**

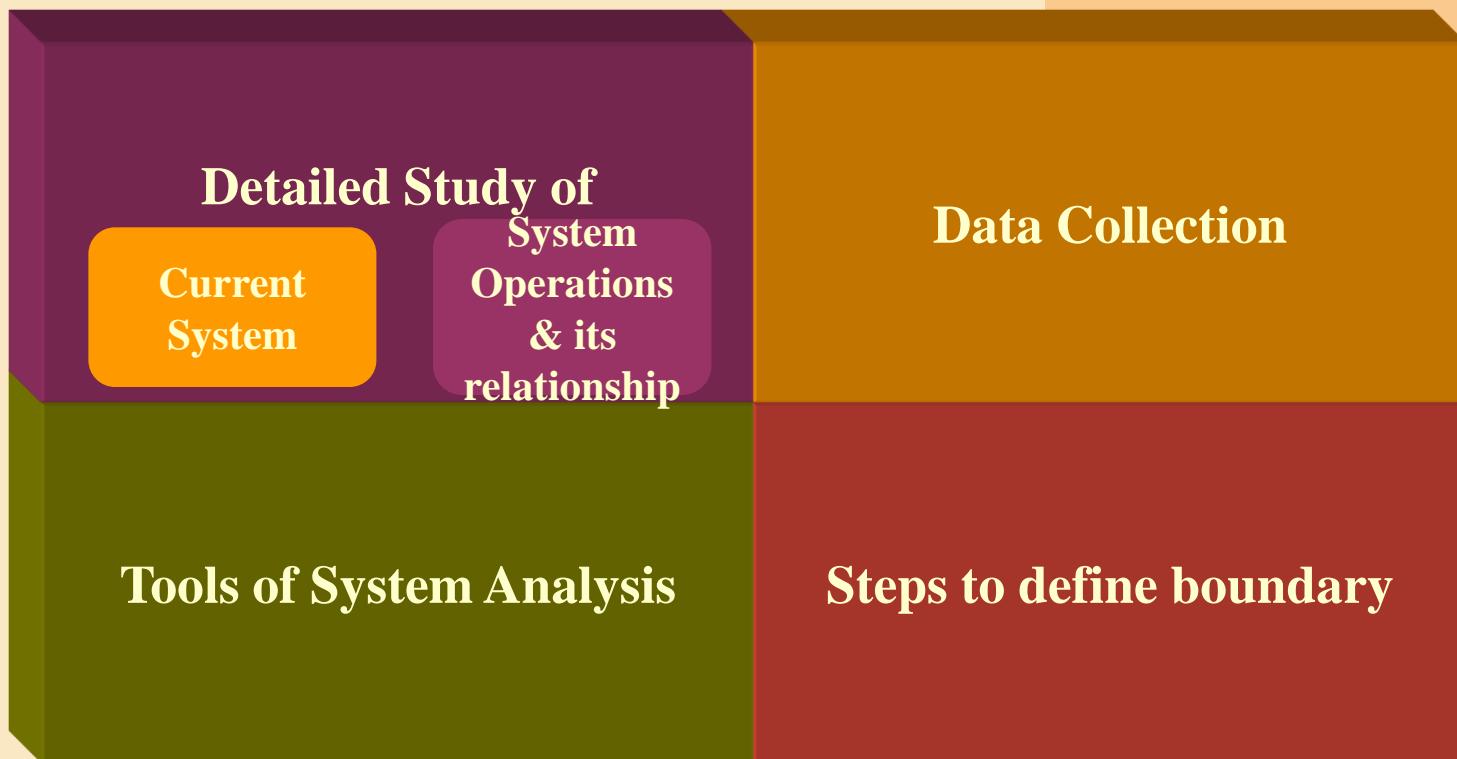
# The System Development Life Cycle

## What is Feasibility?



# The System Development Life Cycle

## What is System Analysis



# The System Development Life Cycle

## What is the Design Phase?

Detailed analysis of a new system.

Major step in moving from problem to solution

Two stages

General  
Design

Detailed  
Design

# The System Development Life Cycle

## What are tools and technique for designing

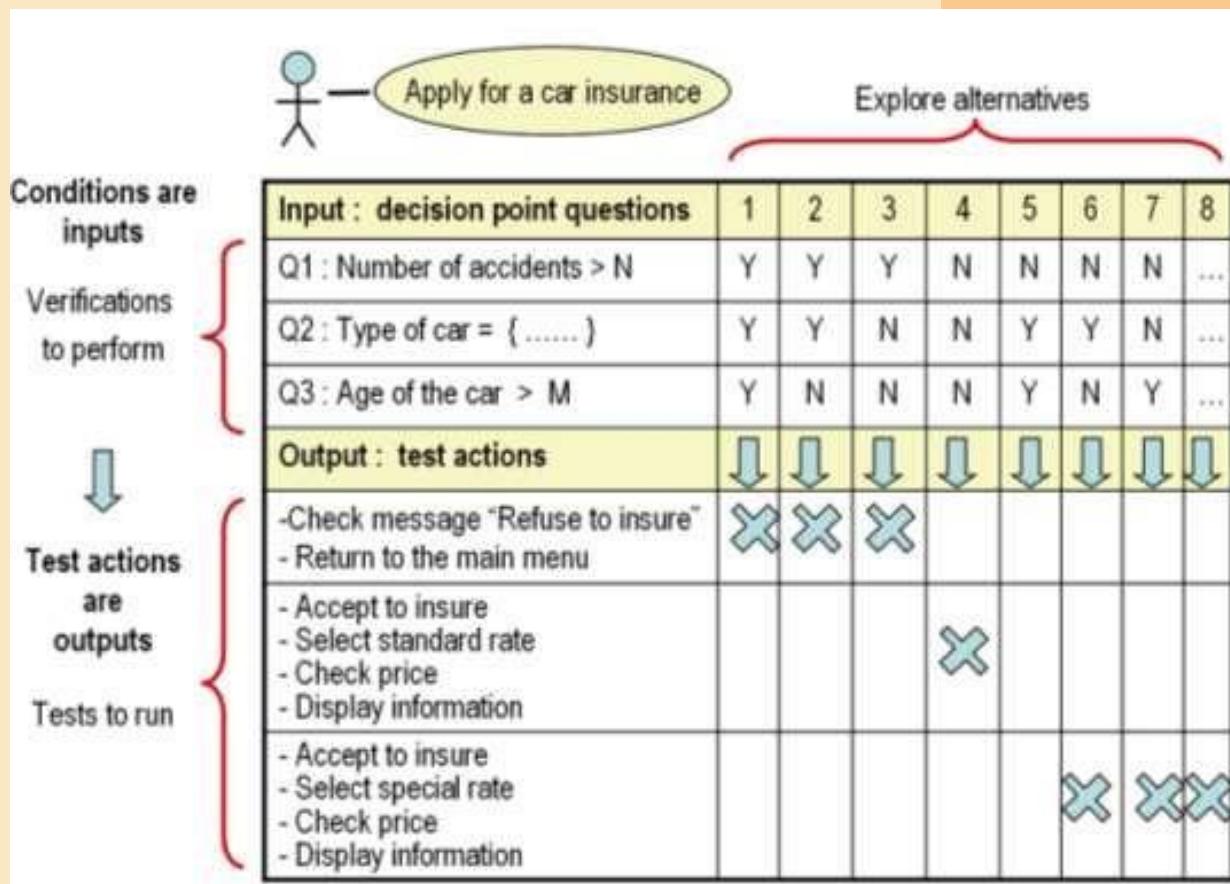
1. Flow Chart

2. Data Flow Diagram

3. Data Dictionary

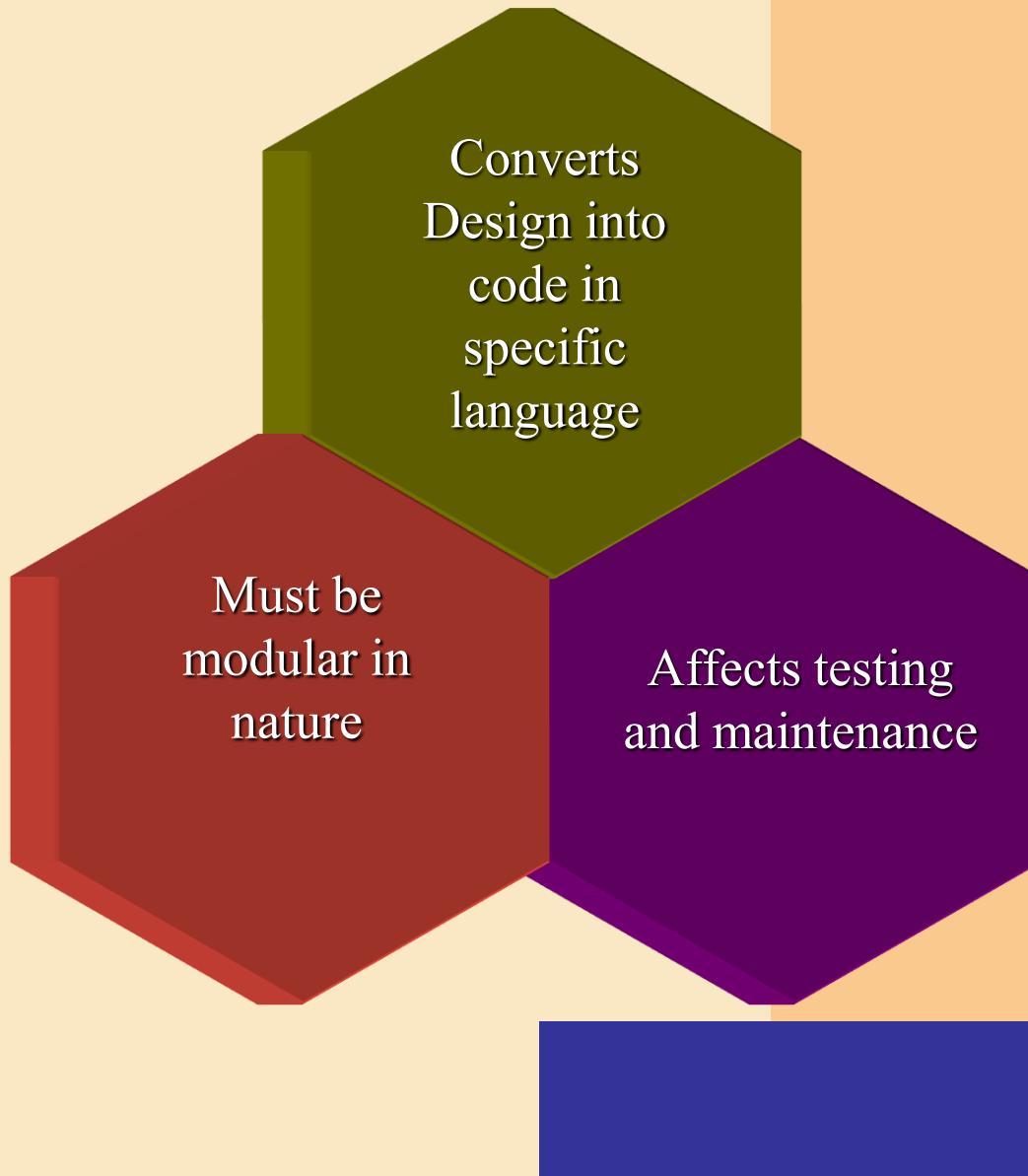
4. Structured English

5. Decision Tables



# The System Development Life Cycle

## What is the Code?



# The System Development Life Cycle

## What is Testing?

Analyze software for finding bugs.

Test plan is developed and run on given set of test data.

# The System Development Life Cycle

## What are different test run?

### Unit Test

Verifies each individual program works by itself

### Systems test

Verifies all programs in application work together

### User Acceptance Testing

determines if the system satisfies the basic requirements

### Integration Test

Verifies application works with other applications

# The System Development Life Cycle

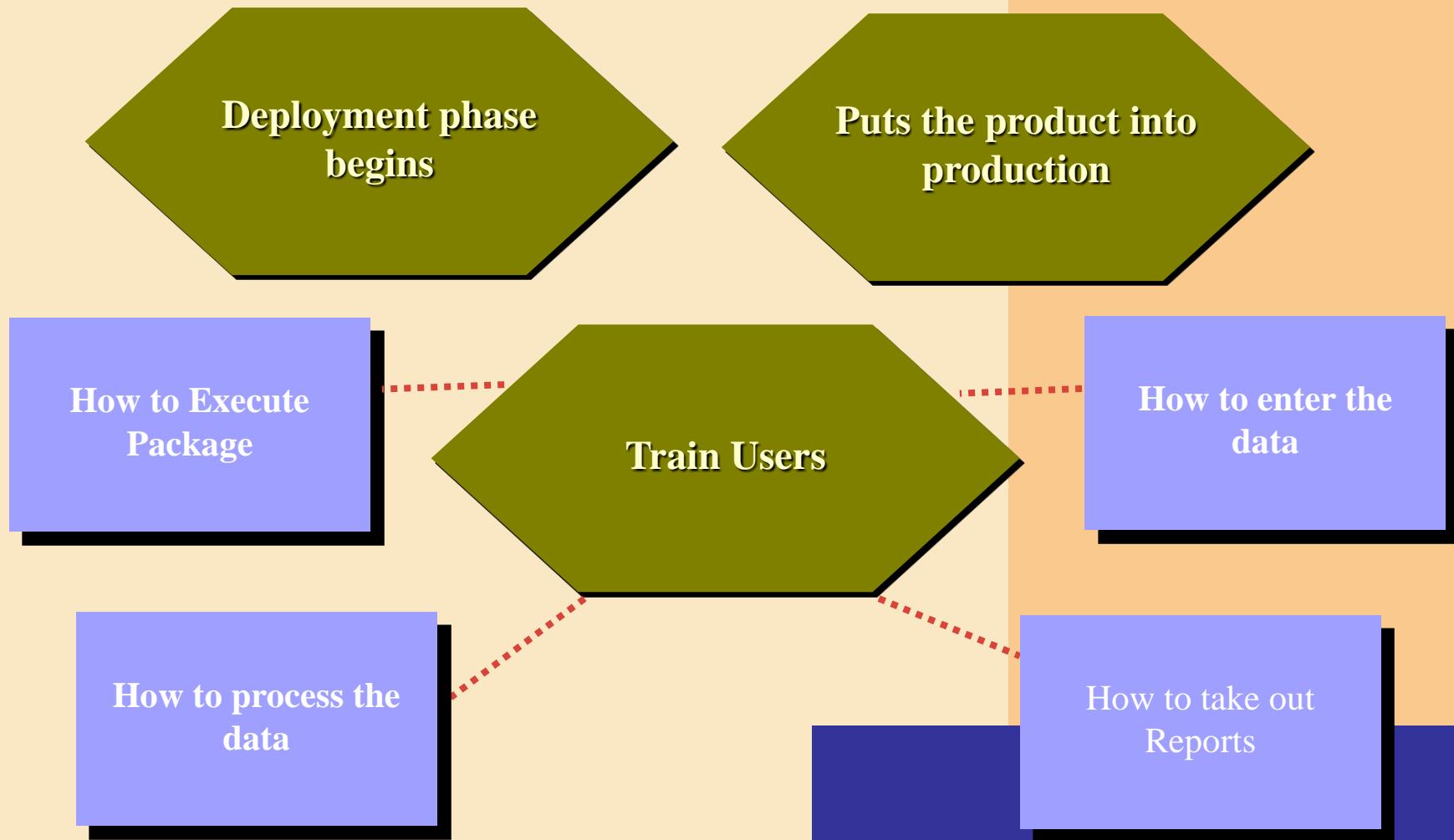
## What is the **implementation?**

- Purpose is to construct, or build, new or modified system and then deliver it to users



# The System Development Life Cycle

## What is Implementation?



# The System Development Life Cycle

## What are implementation strategies?

### System Run Strategies

Parallel Run

Pilot Run

computerized &  
manual systems are  
executed in parallel.

New system is  
installed in parts.

# The System Development Life Cycle

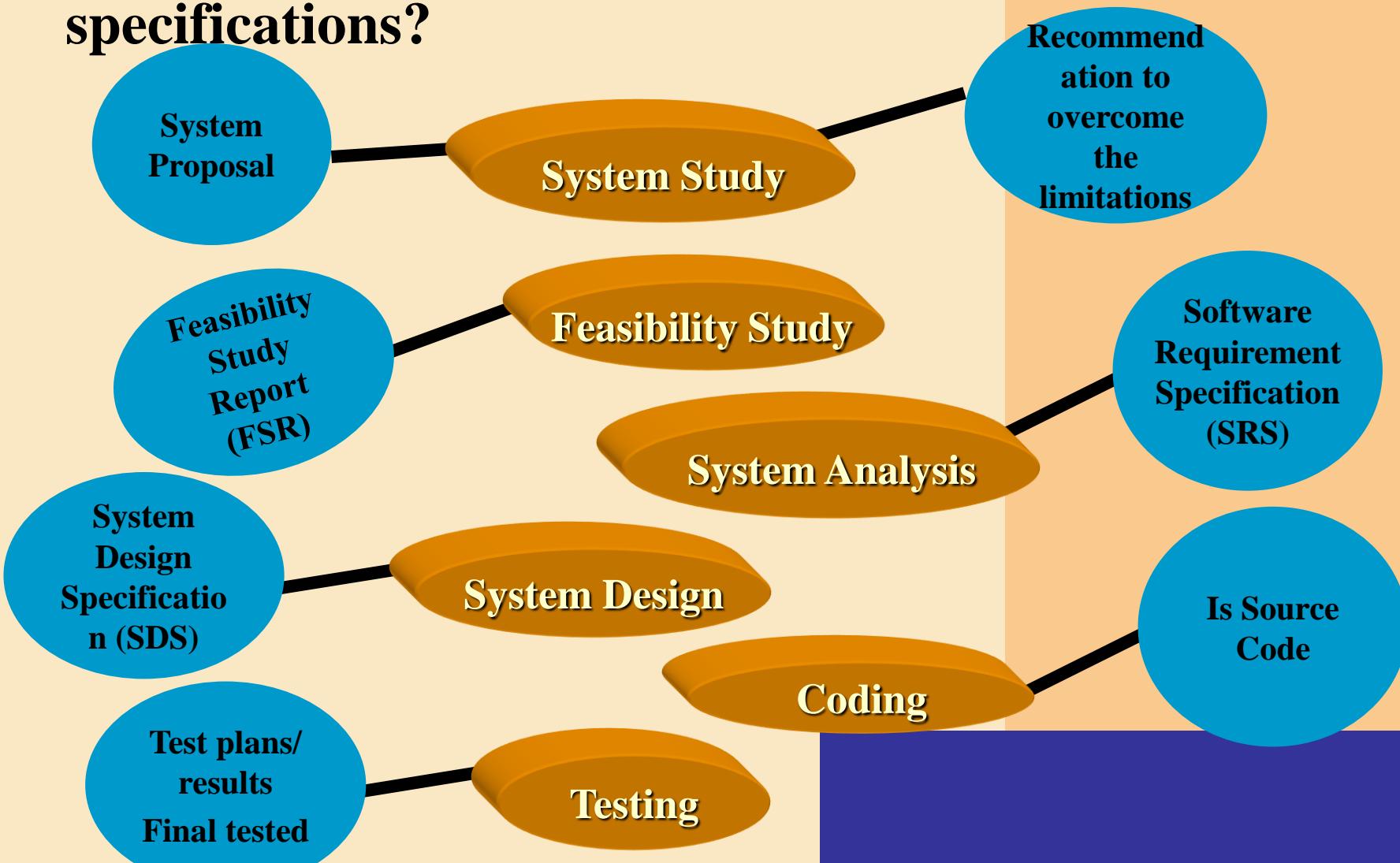
## What is a Maintenance ?

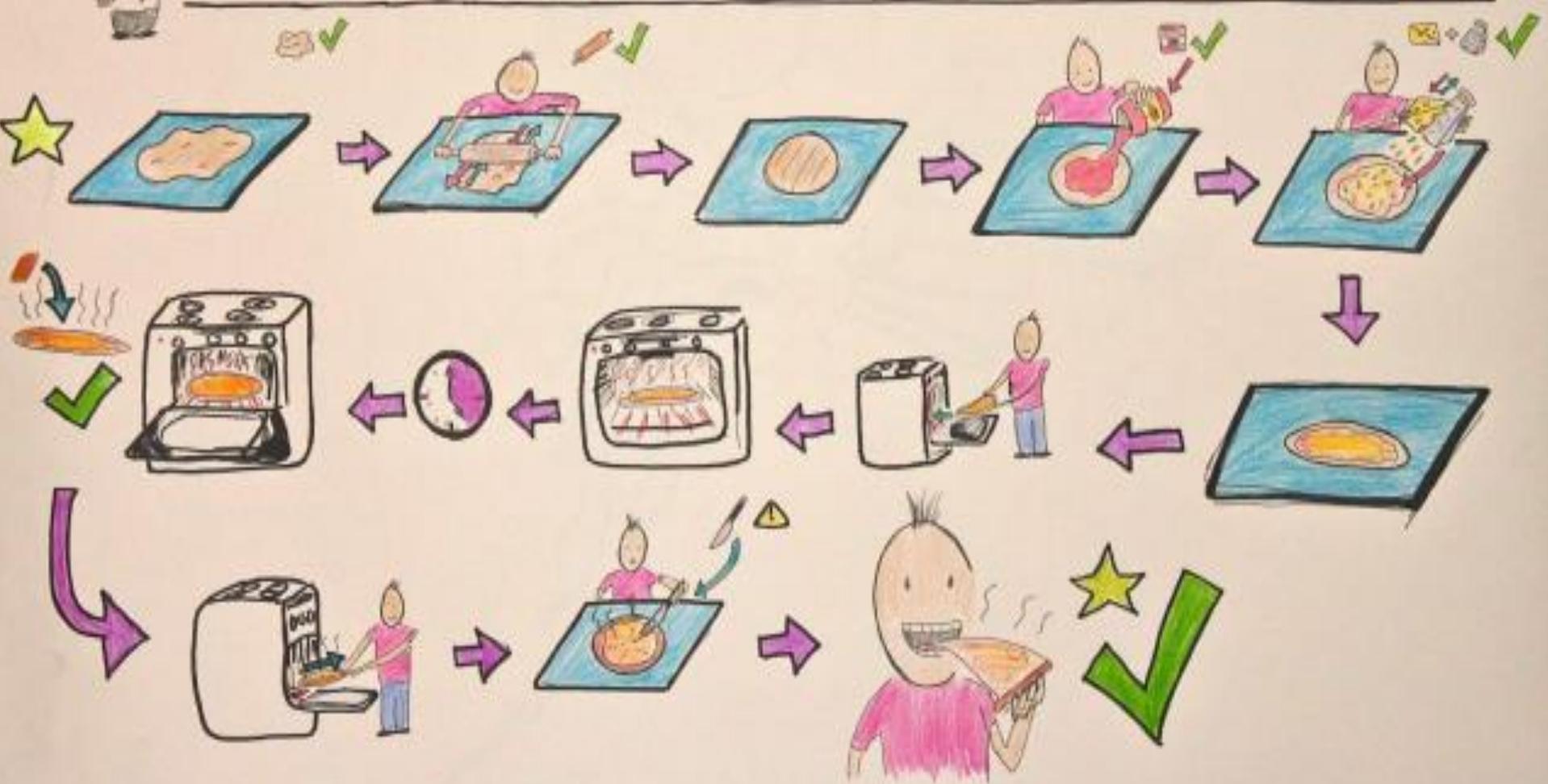
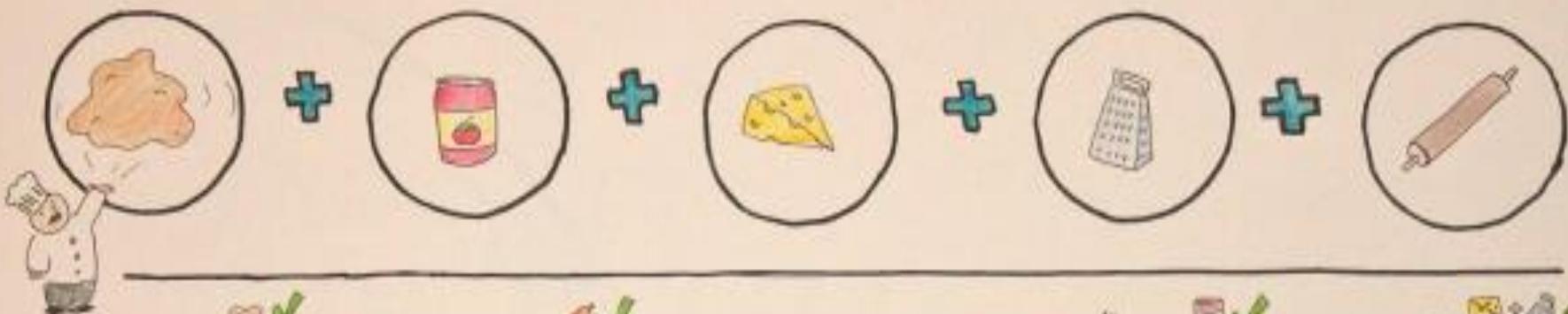
**Eliminate errors in the  
system during its  
working life**

**Tune the system to any validation  
in its working environment**

# The System Development Life Cycle

What are the documents used to summarize technical specifications?





# Home Work

# The System Development Life Cycle

## What are System Environments?

- Development**
- Test**
- Staging**
- Pre-Production**
- Production**
- Mirror**

# *Domain Model*

Lecture 3

## ***Introduction***

---

- A ***domain model*** is the ***most important*** and ***classic-model*** in Object Oriented Analysis

***YOUR ATTENTION PLEASE***

---



# **What is a Domain Model?**

---

- **Problem domain** : area (**scope**) of application that needed to be investigated to solve the problem
- **Domain Model** : Illustrates meaningful conceptual objects in problem domain.
- So domain model are conceptual objects of the area of application to be investigated

# Domain model representation?

- A domain model is a visual representation of **real world concepts** (real-situation objects ), that could be : **idea, thing , event or object.....etc .**
  - **Business objects** - represent things that are manipulated in the business e.g. **Order**.
  - **Real world objects** – things that the business keeps track of e.g. **Contact , book**.
  - **Events** that come to light - e.g. **sale, loan** and **payment**.

# **Domain model representation?**

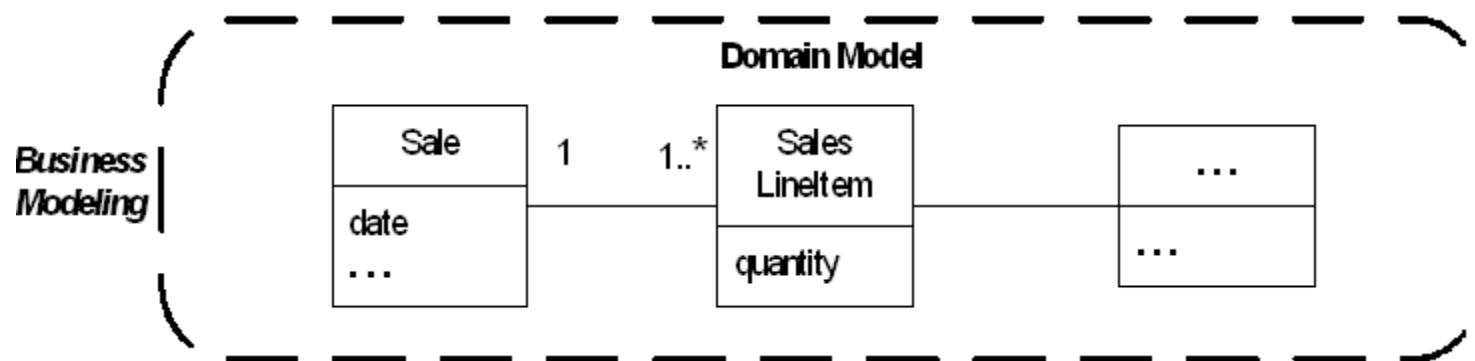
---

***Domain Model may contain :***

- Domain **objects** (conceptual classes)
- **Attributes** of domain objects
- **Associations** between domain objects
- **Multiplicity**

# **Domain Model - UML Notation**

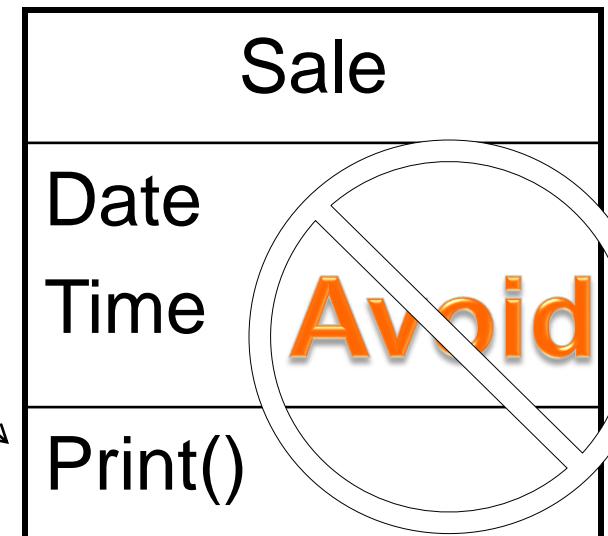
- Illustrated using a set of domain objects (conceptual classes) **with no operations** (**no responsibility assigned yet , this will be assigned during design**).



# A Domain Model is not a Software document

---

Object **responsibilities** is  
not part of the domain  
model. (*But to consider  
during Design*)



# Symbol, intension and extension.



concept's symbol

"A sale represents the event of a purchase transaction. It has a date and time."

concept's intension

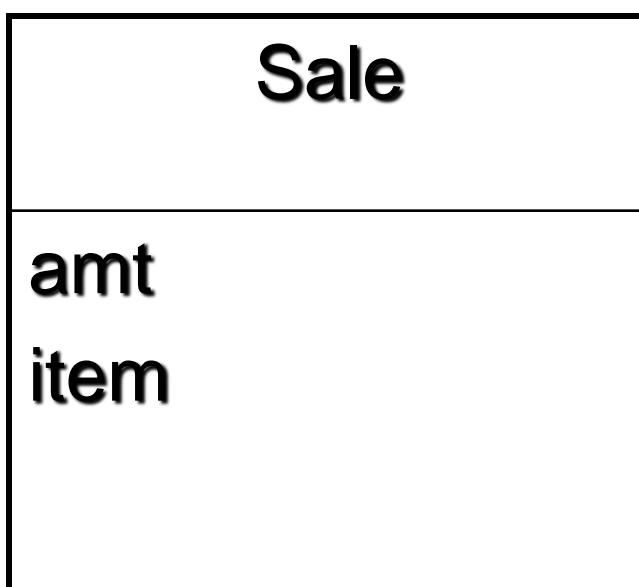
All the examples of sales called **instances**

sale-1  
sale-3  
sale-2  
sale-4

concept's extension

# A Domain Model is Conceptual, not a Software Artifact

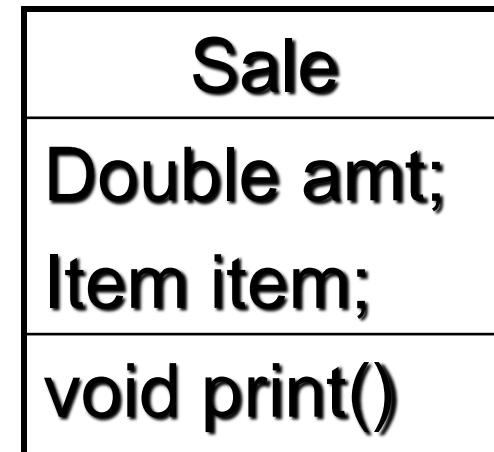
Conceptual Class:



Software Artifacts:



vs.



What's the  
difference?

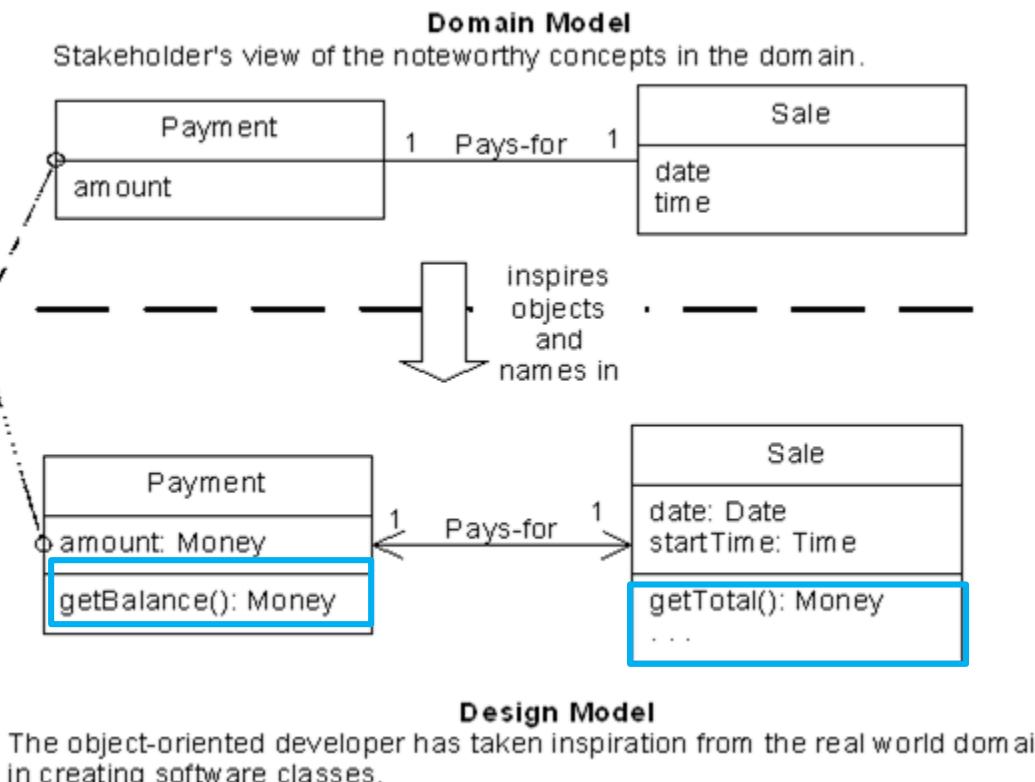
## Question : Why create Domain model ?

## Answer : Get inspiration to create software classes

A Payment in the Domain Model is a concept, but a Payment in the Design Model is a software class. They are not the same thing, but the former *inspired* the naming and definition of the latter.

This reduces the representational gap.

This is one of the big ideas in object technology.



We assign responsibilities during design

***How to find these  
conceptual classes  
and attributes ?***

## ***Method1: Noun Phrase Identification***

---

- *Identify Nouns and Noun Phrases in textual descriptions of the domain that could be :*
  - *The problem definition.*
  - *The Scope.*
  - *The vision.*

## **Method1: Noun Phrase Identification**

---

*However,*

- *Words may be ambiguous ( such as : System )*
- *Different phrases may represent the same concepts.*
- *Noun phrases may also be attributes or parameters rather than classes:*
  - *If it stores state information or it has multiple behaviors, then it's a class*
  - *If it's just a number or a string, then it's probably an attribute*

# Noun Phrase Identification

---

- Consider the following problem description, analyzed for Subjects, Verbs, Objects:

The ATM verifies whether the customer's card number and PIN are correct.

If it is, then the customer can check the account balance, deposit cash, and withdraw cash.

Checking the balance simply displays the account balance.

Depositing asks the customer to enter the amount, then updates the account balance.

Withdraw cash asks the customer for the amount to withdraw; if the account has enough cash, the account balance is updated. The ATM prints the customer's account balance on a receipt.

# Noun Phrase Identification

- Consider the following problem description, analyzed for Subjects, Verbs, Objects:

The ATM verifies whether the customer's card number and PIN are correct.

S      V                    O      O      O

If it is, then the customer can check the account balance, deposit cash, and withdraw cash.

S      V                    O      V      O      V      O

Checking the balance simply displays the account balance.

S      O      V      O

Depositing asks the customer to enter the amount, then updates the account balance.

S      V      O      V      O      V      O

Withdraw cash asks the customer for the amount to withdraw; if the account has enough cash,

S      O      V      O      O      V      S      V      O

the account balance is updated. The ATM prints the customer's account balance on a receipt.

O      V      S      V      O      O

# Noun Phrase

---

Analyze each **subject** and **object** as follows:

- Does it represent a person performing an action? Then it's an actor, '**R**'.
- Is it also a verb (such as 'deposit')? Then it may be a method, '**M**'.
- Is it a simple value, such as 'color' (string) or 'money' (number)?

Then it is probably an attribute, '**A**'.

- Which NPs are unmarked? Make it '**C**' for class.

Verbs can also be classes, for example:

- **Deposit** is a class if it retains state information

# Noun Phrase Identification

- Consider the following problem description, analyzed for Subjects, Verbs, Objects:

The ATM verifies whether the customer's card number and PIN are correct.

S V R A A

If it is, then the customer can check the account balance, deposit cash, and withdraw cash.

R V A V A V V A

Checking the balance simply displays the account balance.

M A V A

Depositing asks the customer to enter the amount, then updates the account balance.

M V R V A M A

Withdraw cash asks the customer for the amount to withdraw; if the account has enough cash,

M A V R A V S V A

the account balance is updated. The ATM prints the customer's account balance on a receipt.

A V S V A A

### **Main Success Scenario (or Basic Flow) of POS:**

---

1. Customer arrives at POS checkout with goods and/or services to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. System records sale line item and presents item description, price, and running total.

Price calculated from a set of price rules.

*Cashier repeats steps 3-4 until indicates done.*

5. System presents total with taxes calculated.
6. Cashier tells Customer the total, and asks for payment.
7. Customer pays and System handles payment.

- 
8. System logs completed sale and sends sale and payment information to the external accounting system (for accounting and commissions) and Inventory system (to update inventory).
  9. System presents receipt.
  10. Customer leaves with receipt and goods (if any).

#### Extensions (or alternative Flows)

- 7a. Paying by cash:
  1. Cashier enters the cash amount tendered.
  2. System presents the balance due, and releases the cash drawer.
  3. Cashier deposits cash tendered and returns balance in cash to Customer.
  4. System records the cash payment.

# Class of POS

Register

Item

Store

Sale

Sales  
LineItem

Cashier

Customer

Ledger

Cash  
Payment

Product  
Catalog

Product  
Description

***Important: There is no such things as a correct list. Brainstorm what you consider noteworthy to be a candidate. But in general , different modelers will find similar lists.***

## **Identification of conceptual classes**

---

- *Identify candidate conceptual classes*
- *Go through them and :*
  - *Exclude irrelevant features and duplications*
  - *Do not add things that are outside the scope ( outside the application area of investigation)*

## ***Method2 : By Category List***

---

***Common Candidates for classes include:***

- *Descriptions , Roles , Places , Transactions*
- *Containers , Systems , abstract nouns , Rules*
- *Organizations, Events, Processes, catalogs , Records , services.*

## ***Attributes***

---

- A *logical data value of an object.*
- *Imply a need to remember information.*
  - Sale needs a **dateTime** attributte
  - Store needs a **name** and **address**
  - Cashier needs an **ID**

## A Common Mistake when modeling the domain- Classes or Attributes?

---

### Rule

- If we do **not think of a thing as a number or text in the real world, then it is probably a **conceptual class**.**
- If it **takes up space**, then it is likely a **conceptual class**.

### Examples:

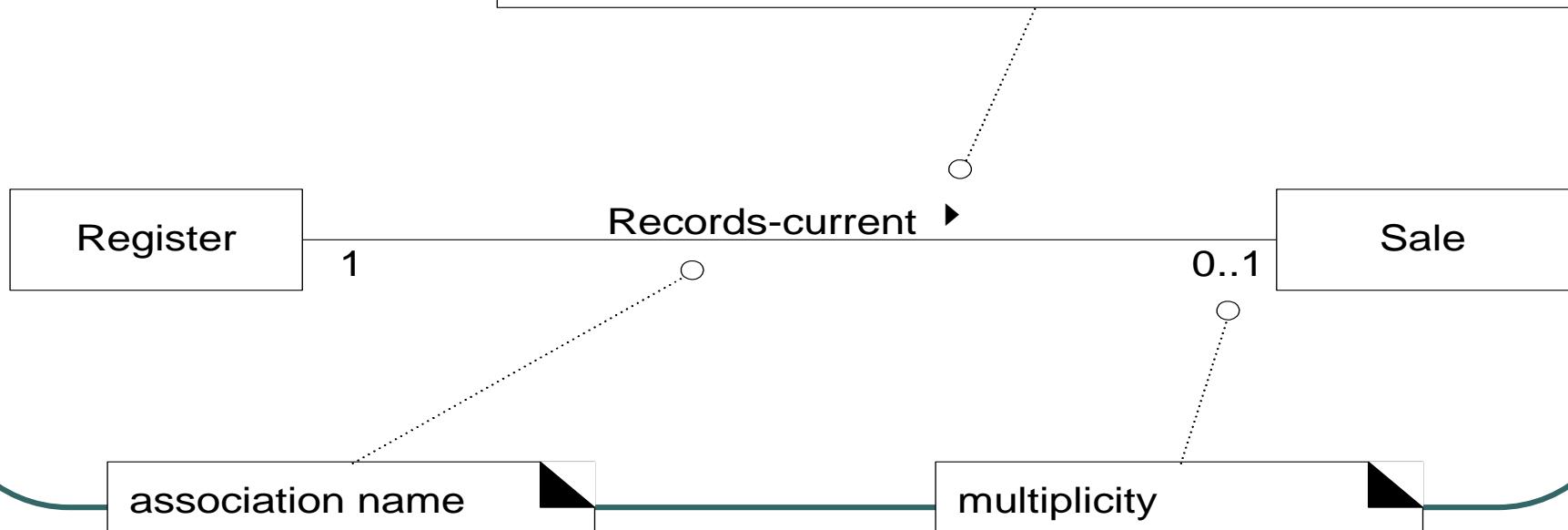
- Is a **store** an attribute of a **Sale** ?
- Is a **destination** an attribute of a **flight** ?

***How to find these  
associations and  
multiplicities?***

## Association:

*Relationship between classes (more precisely, between instances of those classes) indicating some meaningful and interesting connection)*

- "reading direction arrow"
- it has **no** meaning except to indicate direction of reading the association label
- often excluded



## *Common association list*

---

- A is a physical part of B .
  - Wing - Airplane
- A is a logical part of B
  - SalesLineItem - Sale
- A physical contained in B
  - Register-Sale

## *Common association list*

---

- A is a logical contained in B
  - ItemDescription - Catalog
- A is a description of B .
  - ItemDescription - Item
- A is a member of B
  - Cashier – Store

## *Common association list*

---

- A **uses or manage** B
  - Cashier-Register
- A is **recorded in** B
  - Sale-Register
- A is **an organization subunit** of B .
  - Departement - Store

## *Common association list*

---

- A **communicate with** B
  - Customer - Cashier
- A **is related to a transaction** B
  - Customer - Payment
- A **is a transaction related to another transaction** B .
  - Payment - Sale

## *Common association list*

---

- A **is owned by** B
  - Cashier - Register
- A **is an event related** to B
  - Sale- Customer

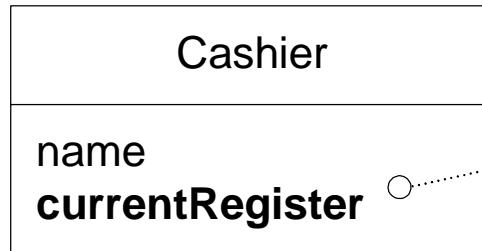
## ***High priority association***

---

- A is a physical or logical part of B
- A is physically or logically contained in/on B
- A is recorded in B
- To avoid:
  - ***Avoid showing redundant associations***
  - ***Do not overwhelm the domain model with associations not strongly required***

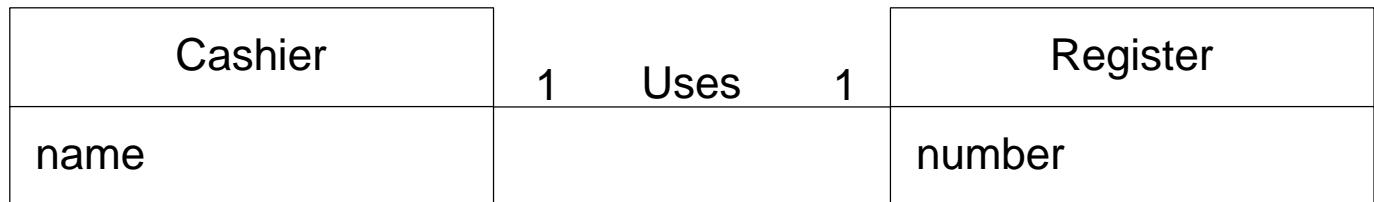
## **Association or attribute ?**

**Worse**



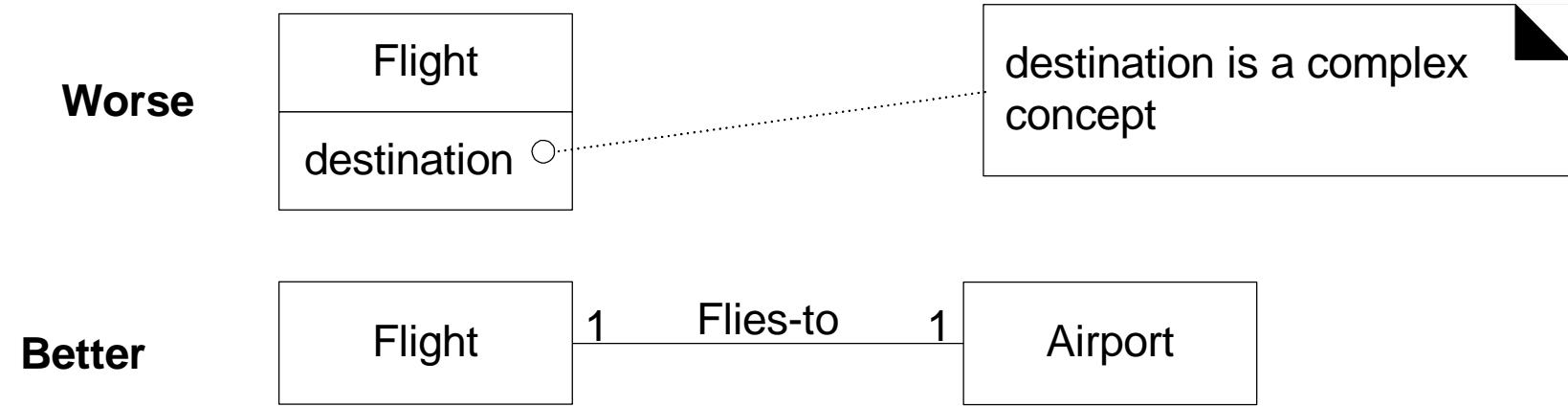
not a "data type" attribute

**Better**



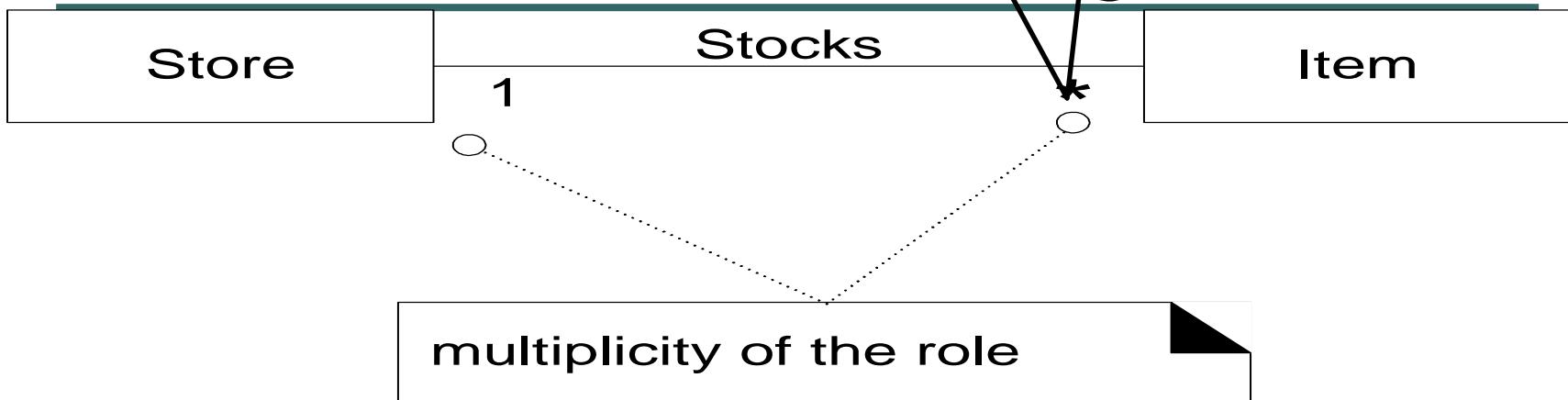
- Most attribute type should be “**primitive**” data type, such as: **numbers** , **string** or **boolean** (true or false)
- Attribute should not be a complex domain concept(Sale , Airport)
- CurrentRegister is of type “Register”, so expressed with an association

## **Association or attribute ?**



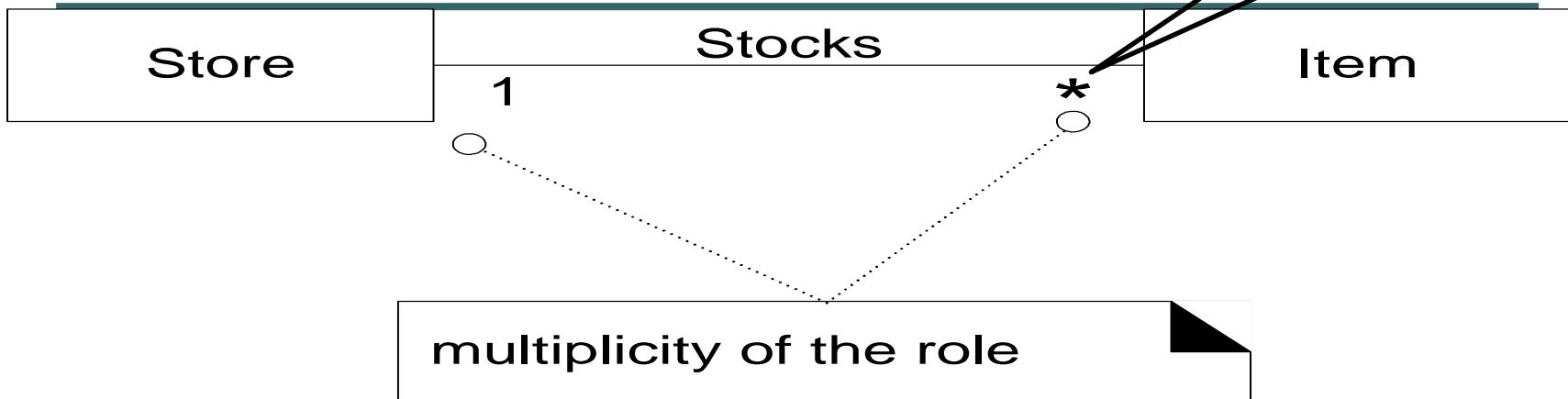
A destination airport is ***not a string***, it is a complex thing that occupies many square kilometers of space. So “Airport” should be related to “Flight” via an association , not with attribute

## Multiplicity



- *Multiplicity indicates how **many instances** can be validly associated with another instance, at a particular moment, rather than over a span of time.*

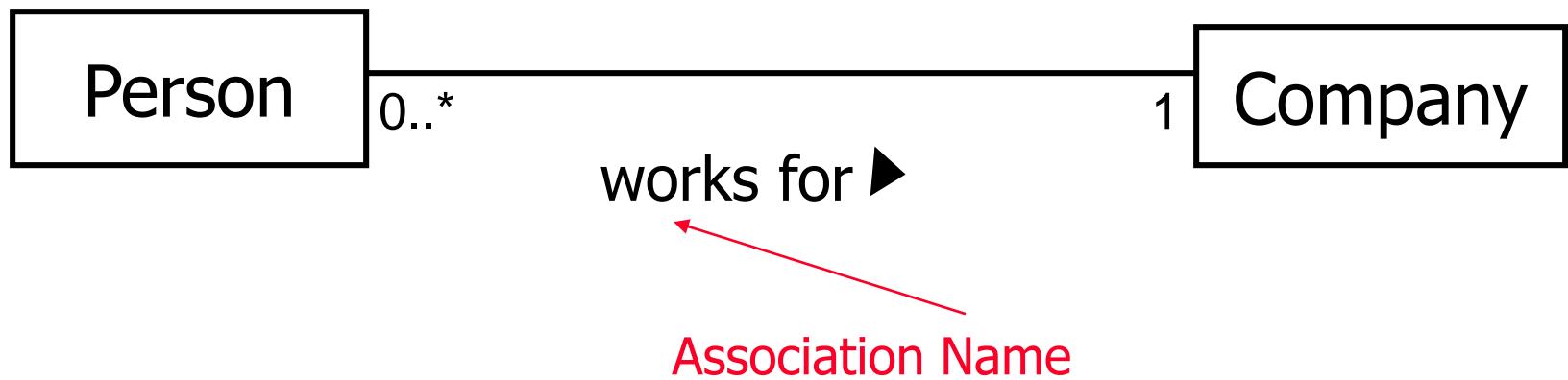
## How to determine multiplicity ?



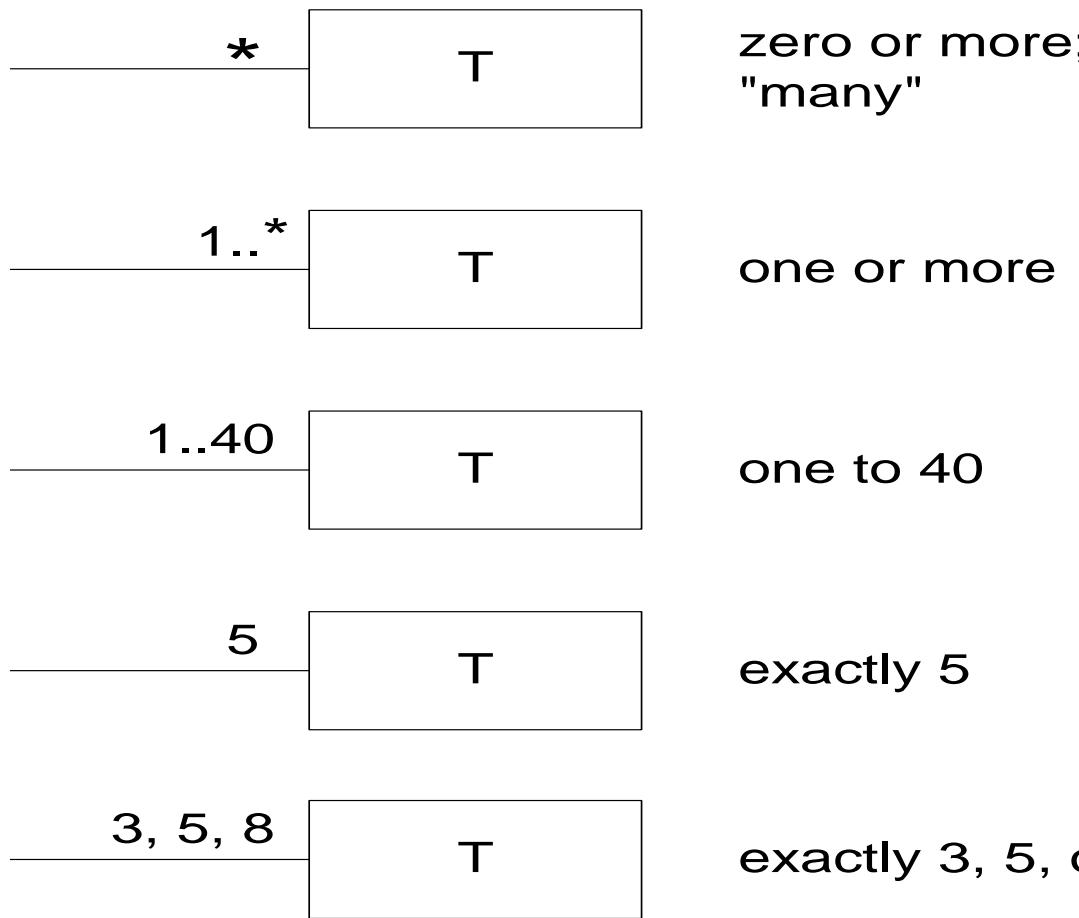
- Ask these 2 questions :
  - *1 store may stock how many item ?*
  - *1 item may be stocked in how many stores ?*

## How to determine multiplicity ?

---



# Multiplicity



## *How to create a domain model*

---

- Identify candidate conceptual classes
- Go through them
  - ***Exclude irrelevant features and duplications***
  - ***Do not add things that are outside the scope***
- Draw them as classes in a UML class diagram
- Add **associations** necessary to record the relationship that must be retained
- Add **attributes** necessary for information to be preserved

## *But remember*

---

- *There is no such thing as a single correct domain model. All models are **approximations of the domain** we are attempting to understand.*
- *We **incrementally evolve a domain model** over several iterations on attempts to capture all possible conceptual classes and relationships.*

# **Case Study (User Story)**

---

- Customer confirms items in shopping cart. Customer provides payment and address to process sale. System validates payment and responds by confirming order, and provides order number that Customer can use to check on order status. System will send customers copy order details by email.
- ***Perform the steps of Domain model on the mentioned case study.***

# Remove Irrelevant Data

---

Customer

Item

Shopping Cart

Payment

~~Address~~

~~Sale~~

Order

~~Order Number~~

~~Order Status~~

~~Order Details~~

~~Email~~

~~System~~

# Finding Conceptual Classes

---

Customer

Shopping Cart

Payment

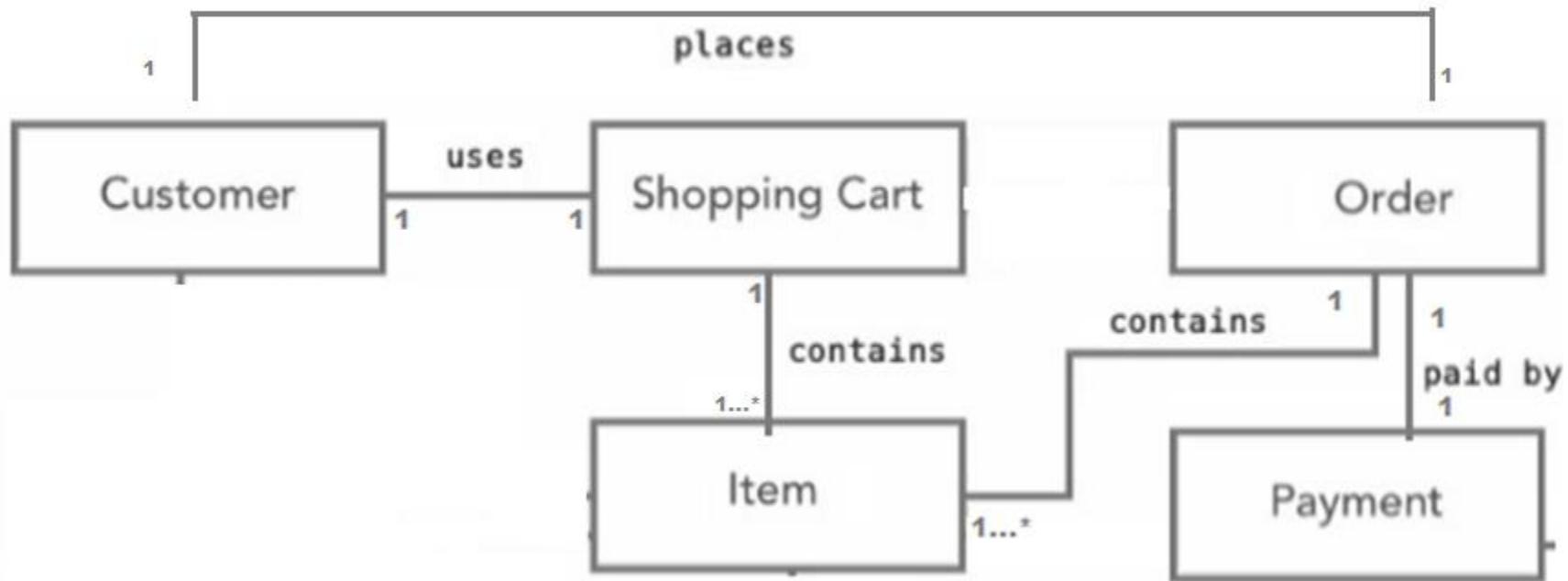
Item

Order

# Add Association



# Add Multiplicity



## ***Homework: Example Restaurant case***

---

### *Record Booking*

- Receptionist enters date of requested reservation;
- System displays bookings for that date;
- There is a suitable table available: Receptionist enters details (customer's name, phone number, time of booking, number of covers, table number);
- System records and displays new booking.

# Use Cases

- One of the primary challenges is the ability to elicit the correct and necessary system requirements from the stakeholders and specify them in a manner understandable to them so those requirements can be verified and validated.

*The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines, and to other software systems. No other work so cripples the resulting system if done wrong. No other part is more difficult to rectify later.*

Fred Brooks

# Use Case Diagram – Guidelines & Caution

1. Use cases should ideally begin with a verb – i.e. generate report. Use cases should NOT be open ended – i.e. Register (instead should be named as Register New User)
2. Avoid showing communication between actors.
3. Actors should be named as singular. i.e. student and NOT students. NO names should be used – i.e. John, Sam, etc.
4. Do NOT show behavior in a use case diagram; instead only depict only system functionality.
5. Use case diagram does not show sequence – unlike DFDs.

# System Concepts for Use-Case Modeling

**Use case** – a behaviorally related sequence of steps (scenario), both automated and manual, for the purpose of completing a single task.

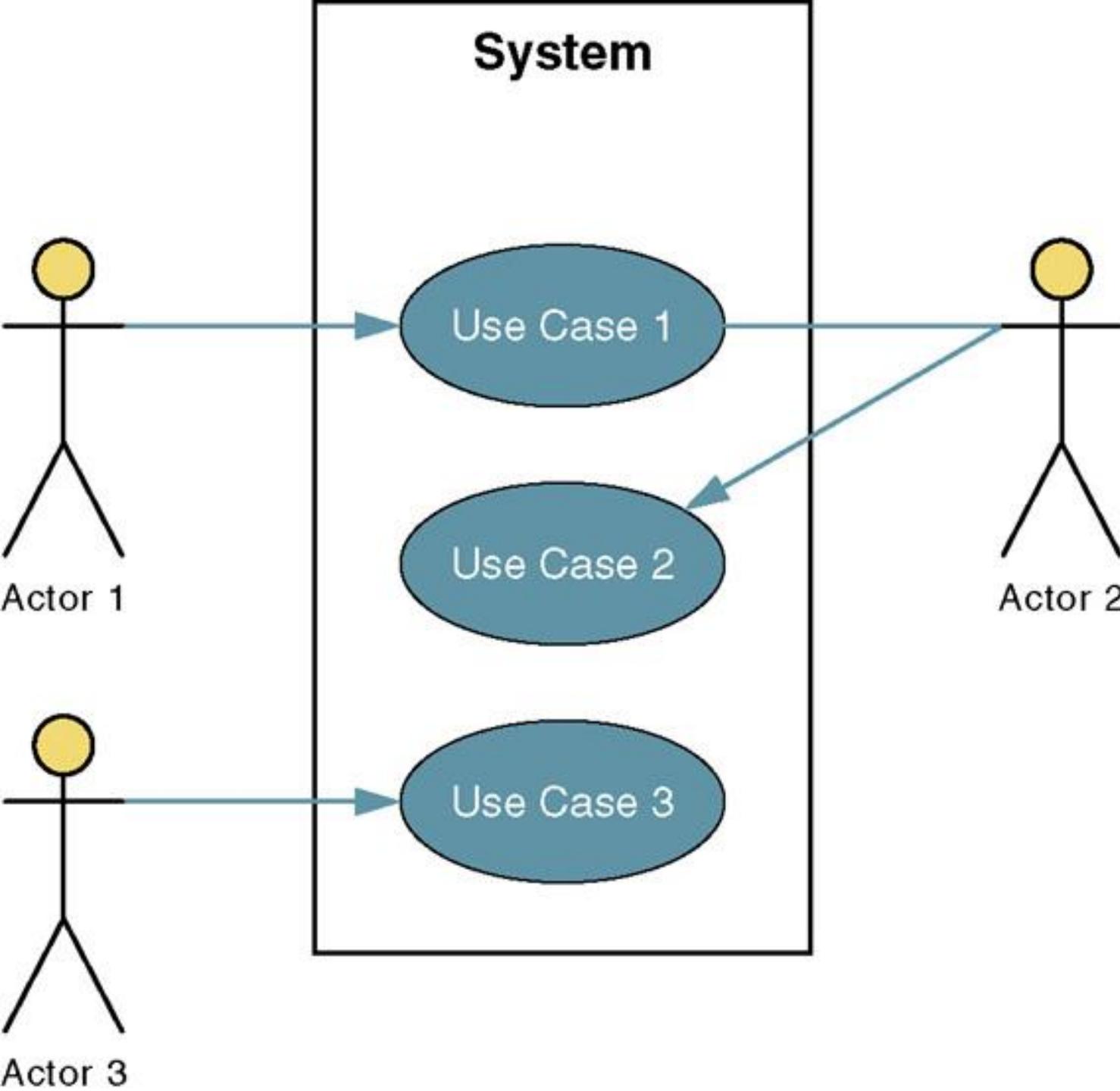
- Description of system functions from the perspective of external users in terminology they understand.

**Use-case diagram** – a diagram that depicts the interactions between the system and external systems and users.

- graphically describes who will use the system and in what ways the user expects to interact with the system.

**Use-case narrative** – a textual description of the event and how the user will interact with the system to accomplish the task.

# Sample Use- Case Model Diagram



# Basic Use-Case Symbols

**Use case** – subset of the overall system functionality

- Represented by a horizontal ellipse with name of use case above, below, or inside the ellipse.

Use Case  
Symbol

**Actor** – anyone or anything that needs to interact with the system to exchange information.

- human, organization, another information system, external device, even time.



Actor Symbol

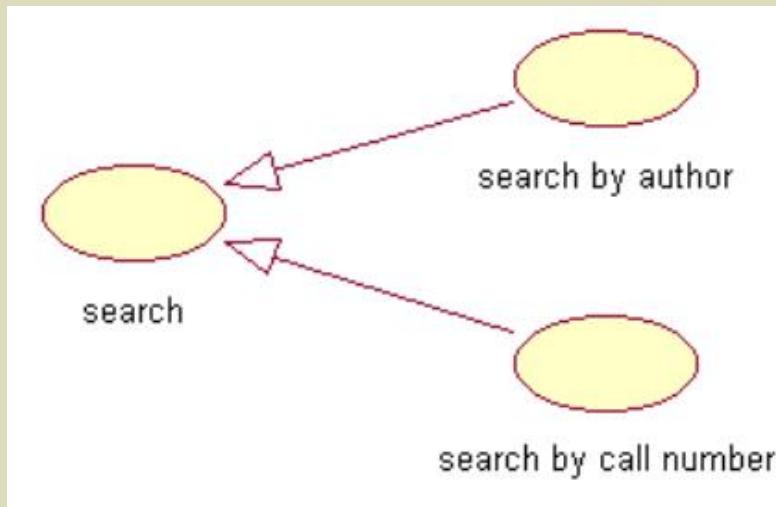
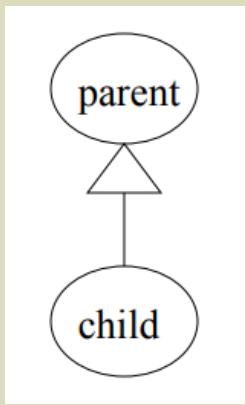
# Use Case - Relationships and its Types

- Association relationship: Represent communication between actor and use case
  - Often referred to as a communicate association
  - use just a line to represent
-

# Use Case - Relationships and its Types

## Generalization:

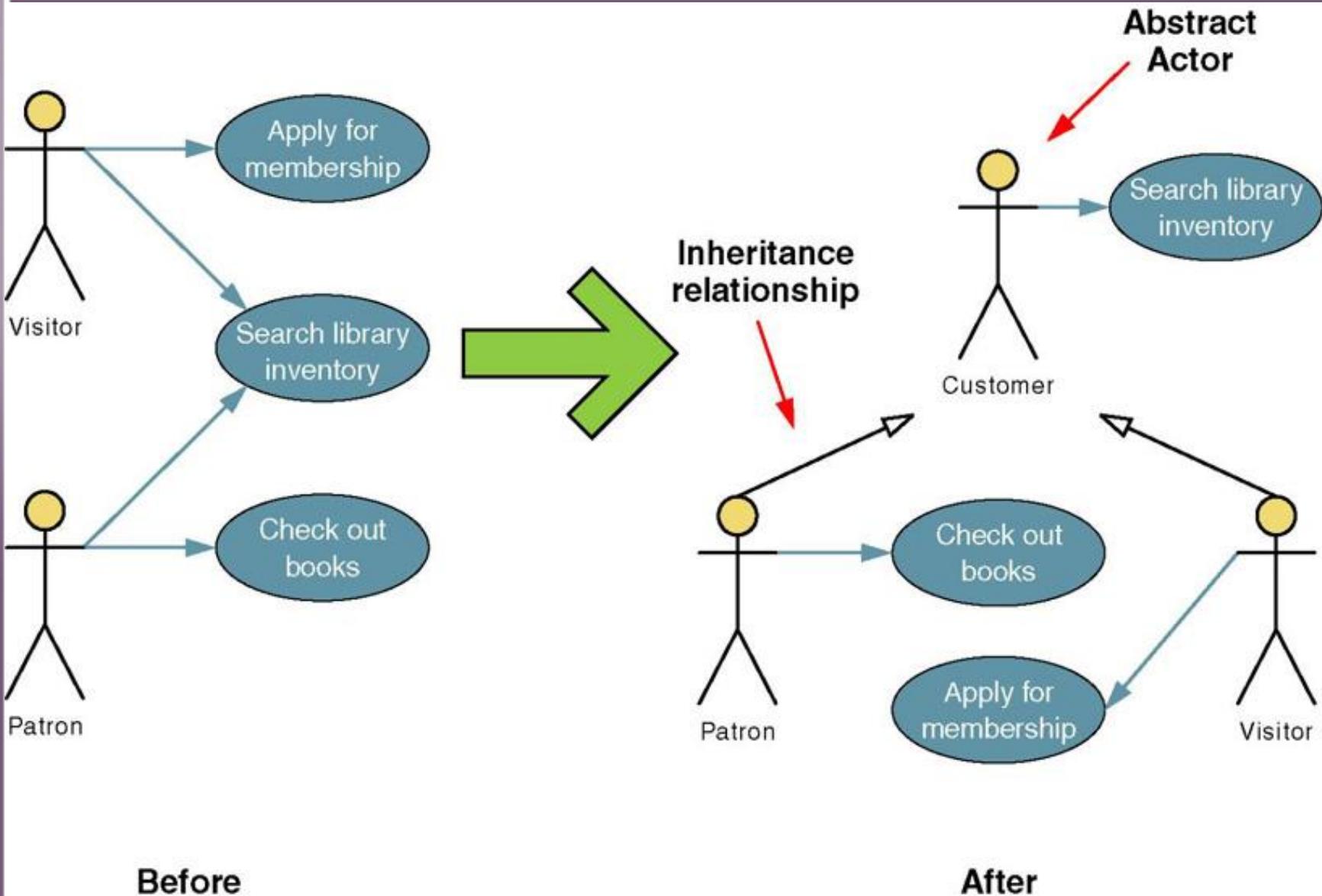
- The child use case inherits the behaviour and meaning of the parent use case.
- The child may add to or override the behaviour of its parent.
- Notation:



# Use Case - Relationships and its Types

- Generalization relationship between actors
  - actor generalization refers to the relationship which can exist between two actors
- Generalization relationship between use cases
  - use case generalization refers to the relationship which can exist between two use cases

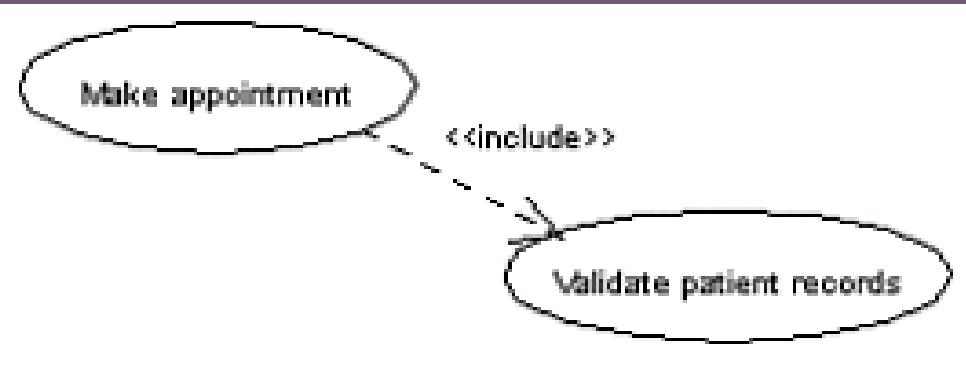
# Use Case Inheritance Relationship



# Use Case - Relationships and its Types

- **Include**
  - Specifies that the source use case explicitly incorporates the behavior of another use case at a location specified by the source
  - The include relationship adds additional functionality not specified in the base use case.
  - <<include>> is used to include common behavior from an included use case into a base use case in order to support re-use of common behavior.
  - Notation      <<include>>,>

# <<include>>



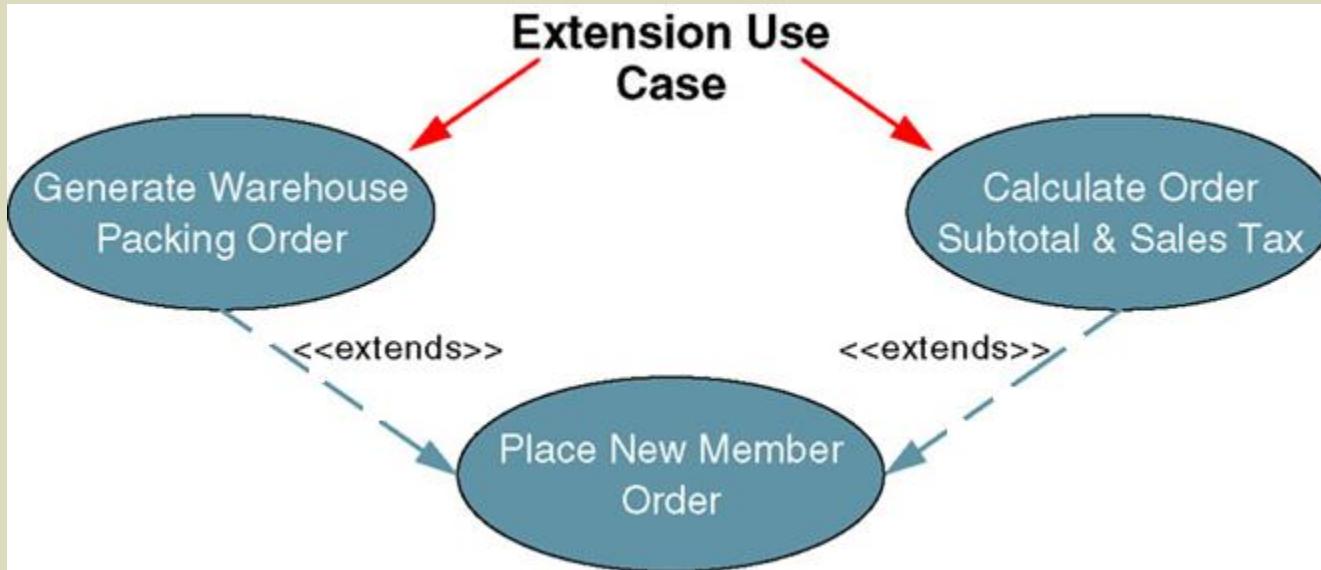
In Figure, the functionality defined by the "Validate patient records" use case is contained within the "Make appointment" use case.

Hence, whenever the "Make appointment" use case executes, the steps defined in the "Validate patient records" use case are also executed.

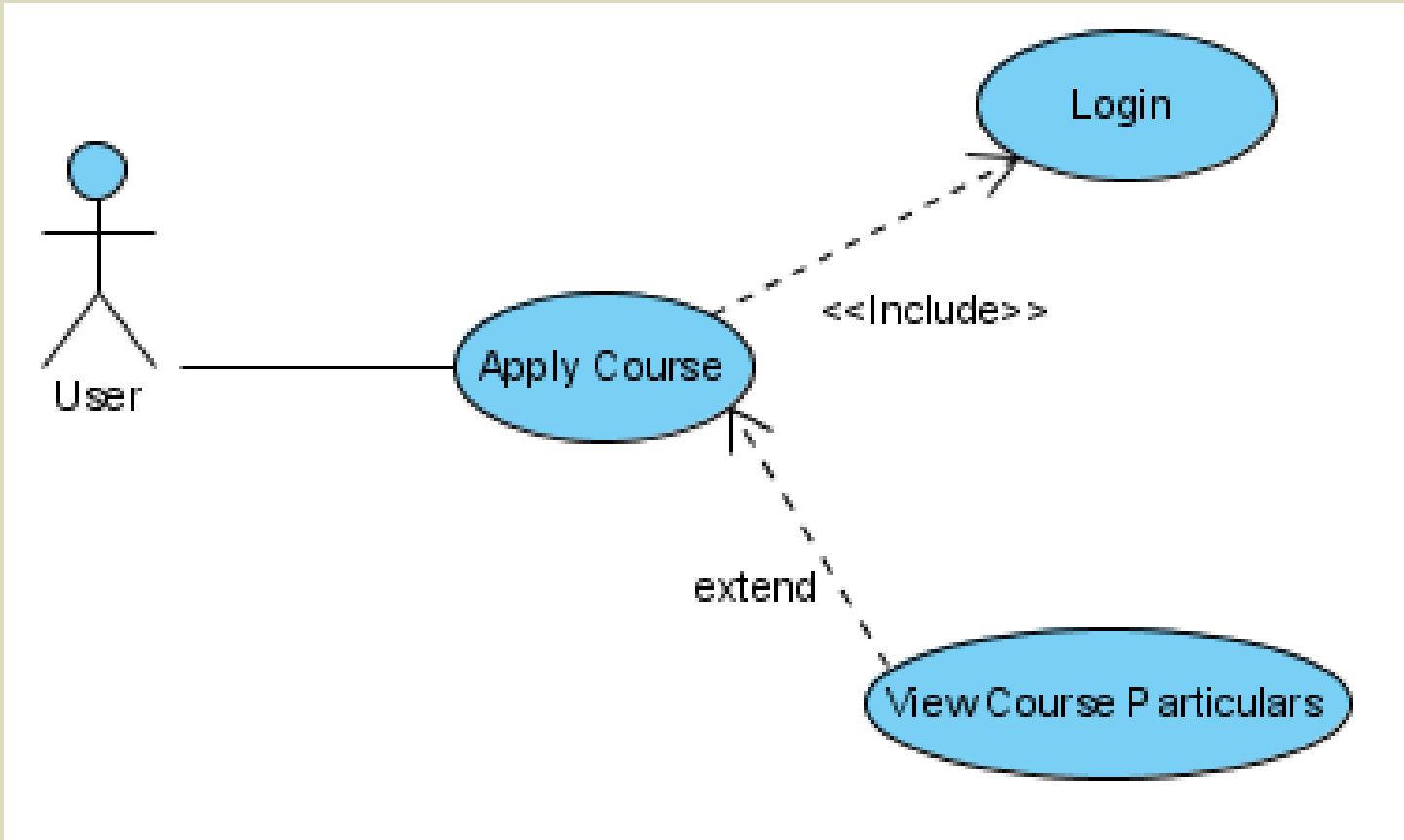
# Use Case - Relationships and its Types

- **Extend**
  - Specifies that the target use case extends the behavior of the source.
  - The extend relationships shows optional functionality or system behavior.
  - <<extend>> is used to include optional behavior from an extending use case in an extended use case.
- Notation 

# <<extend>>



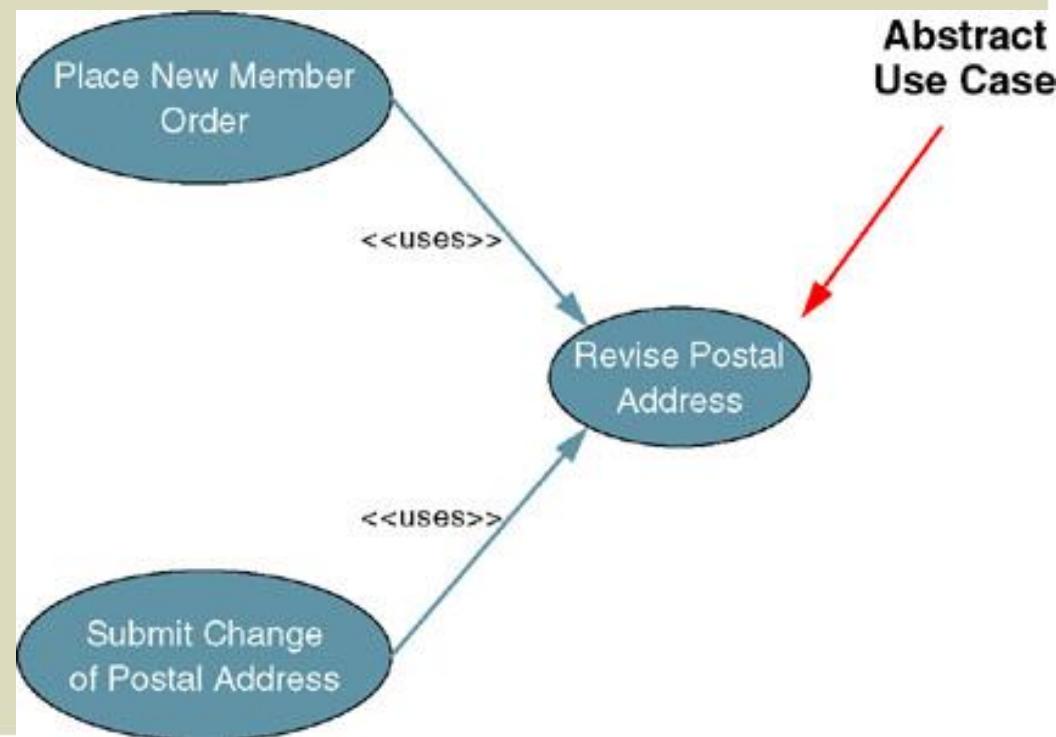
# Example – Include and Extend



# Use Case Uses Relationship

**Abstract use case** – use case that reduces redundancy in two or more other use cases by combining common steps found in both.

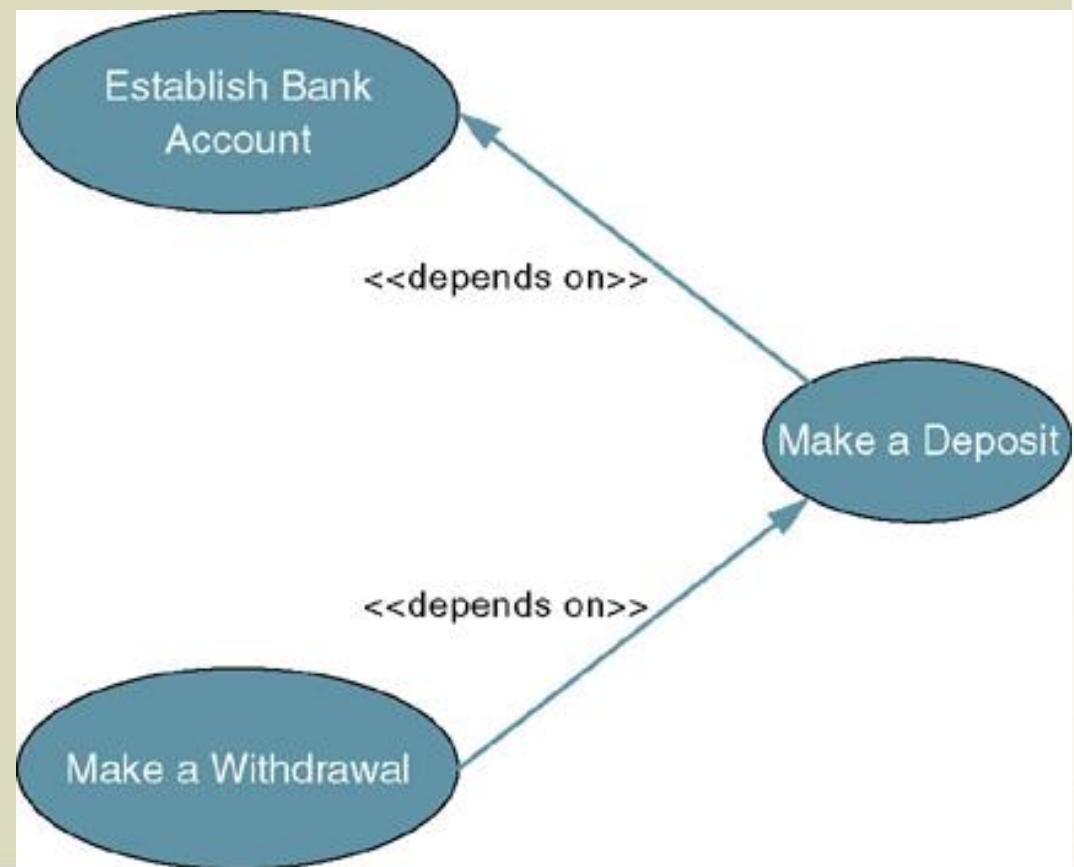
- Available by any other use case that requires its functionality.
- Generally not identified in requirements phase
- Relationship between abstract use case and use case that uses it is called a *uses* (or *includes*) relationship.
- Depicted as arrow beginning at original use case and pointing to use case it is using.
- Labeled <<uses>>.



# Use Case Depends On Relationship

**Depends On** – use case relationship that specifies which other use cases must be performed before the current use case.

- Can help determine sequence in which use cases need to be developed.
- Depicted as arrow beginning at one use case and pointing to use case it depends on.
- Labeled <<depends on>>.



# Use Case - Boundary

## Boundary

- A boundary rectangle is placed around the perimeter of the system to show how the actors communicate with the system.
- A system boundary of a use case diagram defines the limits of the system.

# Questions for Identifying People Actors

- Who is interested in the scenario/system?
- Where in the organization is the scenario/system be used?
- Who will benefit from the use of the scenario/system?
- Who will supply the scenario/system with this information, use this information, and remove this information?
- Does one person play several different roles?
- Do several people play the same role?

# Questions for Identifying Other Actors

- What other entity is interested in the scenario/system?
- What other entity will supply the scenario/system with this information, use this information, and remove this information?
- Does the system use an external resource?
- Does the system interact with a legacy system?

# Courseware Management System

- Course administrators manages topics and courses that make up a course, additionally he also can view courses.
- Course administrators manages Tutor who teach courses
- Course administrators who mange the assignment of the courses to tutors
- Calendar or Course Schedule is generated as a result of the Students who refer to the Course schedule or Calendar to decide which courses for which they wish to take.

# User Placing an Order

- A customer placing an order with a sales company might follow these steps :
  1. Browse catalog and select items.
  2. Call sales representative.
  3. Supply shipping information.
  4. Supply payment information.
  5. Receive conformation number from salesperson.

# Vending Machine

1. A customer buys a product
2. The supplier restocks the machine
3. The supplier collects money from the machine

# Altered State University (ASU) Registration System

1. Professors indicate which courses they will teach on-line.
2. A registrar creates and prints the course catalog.
3. Allows students to select on-line four courses for upcoming semester.
4. No course may have more than 10 students or less than 3 students.
5. When the registration is completed, the system sends information to the billing system.
6. Professors can obtain course rosters on-line.
7. Students can add or drop classes on-line.
8. It is the duty of registrar to maintain the course, professor and student information on-line.

# Vehicles Sale System.

- Draw a use case diagram for the vehicle sales system. Customer makes offer for the vehicle. Customer can be new customer or old customer. New and old customer can make their own offers. For every individual they have to get registered. System can update the existing customer information as well. Customer make payment if his/ her offer is accepted. Management has right to accept or reject the offer by managing the offer. Sales person records the sales contract of the accepted offer.

# Home Work

# Hospital Management System

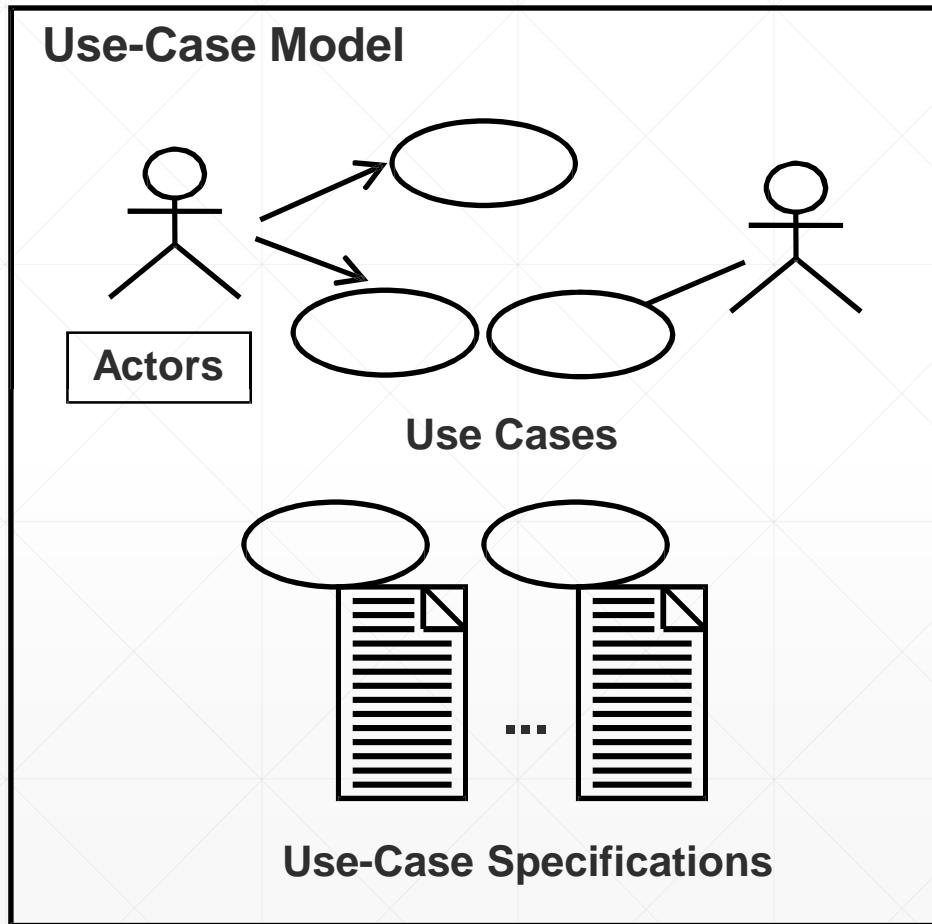
- Draw a use case diagram for the hospital reception system. In this system, receptionist can schedule patient appointment and patient hospital admission after the patient registration. Both types of patients i.e. outpatient and inpatient can be admitted in the hospital. Receptionist also checks the insurance and claim forms and put them in file. Patient medical report is also filed by the receptionist.

# Use Case Model

---

Use case description

# Relevant Requirements Artifacts



**Supplementary Specification**

# Full Use Case Description

- Shows steps (“Flow of Events”) in more detailed and are structured; they dig deeper

# A Recommended Template:

Use Case Description	
Use Case name:	
Use Case Description:	
Primary actor:	Other actors:
Stakeholders:	
Relationships <ul style="list-style-type: none"><li>▪ Includes:</li><li>▪ Extends:</li></ul>	
<b>Pre-conditions:</b> <ul style="list-style-type: none"><li>▪ </li></ul>	
<b>Flow of Events:</b> <ol style="list-style-type: none"><li>1. Actor does....</li><li>3.</li><li>4.</li></ol>	
<b>Alternative and exceptional flows:</b> <ol style="list-style-type: none"><li>4.1 ....</li></ol>	
<b>Post-conditions:</b> <ul style="list-style-type: none"><li>▪ </li></ul>	

# Use Cases Basics

- A use case has four mandatory elements:
  1. **Name:** Each use case has a unique name describing what is achieved by the interaction with the actor.  
EX: "Turn Light On/Off" and "Print Document" are good examples.
  2. **Brief description:** The purpose of the use case should be described in one or two sentences.  
EX: "This use case controls the selected light bank when instructed by the actor Homeowner."
  3. **Actor(s)** List each actor that participates in the use case.
  4. **Flow of events:** The heart of the use case is the event flow, usually a textual description of the interactions between the actor and the system.
    - The main (basic) flow of events
    - The alternate flows of events

# Use Cases Basics

- Optional elements in a Use case:
  - **Pre-conditions**: Must be present in order for a use case to start. Represent some system state that must be present before the use case can be used.

**EX:** A pre-condition of the "Print Author's Manuscript Draft" use case is that a document must be open.

- **Post-conditions**: Describe the state of the system after a use case has run. Represent persistent data that is saved by the system as a result of executing the use case.

**EX:** Post-condition of "Register" use case is that the new data is added in the profile of the student.

# Use Cases Basics

- **Other stakeholders:** Other key stakeholders who may be affected by the use case.  
**EX:** A manager may use a report built by the system, and yet the manager may not personally interact with the system in any way and therefore would not appear as an actor on the system.

# Brief Description of Use Case

## *Create new order description*

When the customer calls to order, the order clerk and system verify customer information, create a new order, add items to the order, verify payment, create the order transaction, and finalize the order.

- Same description that is usually captured in initial Use Case Diagrams

# Full Use Case Description

- Telephone Order Scenario for Create New Order Use Case

<b>Use Case Name:</b>	Create new order
<b>Brief Description:</b>	When customer calls to order, the order clerk and system verify customer information, create a new order, add items to the order, verify payment, create the order transaction, and finalize the order.
<b>Actors:</b>	Telephone sales clerk
<b>Related Use Cases:</b>	Includes: <i>Check item availability</i>
<b>Stakeholders:</b>	Sales department: to provide primary definition Shipping department: to verify that information content is adequate for fulfillment Marketing department: to collect customer statistics for studies of buying patterns
<b>Preconditions:</b>	Customer must exist. Catalog, Products, and Inventory items must exist for requested items.
<b>Postconditions:</b>	Order and order line items must be created. Order transaction must be created for the order payment. Inventory items must have the quantity on hand updated. The order must be related (associated) to a customer.

# Full Use Case Description

- Telephone Order Scenario for Create New Order Use Case

	<p><b>Flow of Events:</b></p> <ol style="list-style-type: none"><li>1. Sales clerk answers telephone and connects to a customer.</li><li>2. Clerk verifies customer information.</li><li>3. Clerk initiates the creation of a new order.</li><li>4. Customer requests an item be added to the order.</li><li>5. Clerk verifies the item (<i>Check item availability use case</i>).</li><li>6. Clerk adds item to the order.</li><li>7. Repeat steps 4, 5, and 6 until all items are added to the order.</li><li>8. Customer indicates end of order; clerk enters end of order.</li> <li>9. Customer submits payment; clerk enters amount.</li></ol>
<b>Exception Conditions:</b>	<p>2.1 If customer does not exist, then the clerk pauses this use case and invokes <i>Maintain customer information</i> use case.</p> <p>2.2 If customer has a credit hold, then clerk transfers the customer to a customer service representative.</p> <p>4.1 If an item is not in stock, then customer can</p> <ol style="list-style-type: none"><li>a. choose not to purchase item, or</li><li>b. request item be added as a back-ordered item.</li></ol> <p>9.1 If customer payment is rejected due to bad-credit verification, then</p> <ol style="list-style-type: none"><li>a. order is canceled, or</li><li>b. order is put on hold until check is received.</li></ol>

# Use-Cases – Common Mistakes

- Complex diagram
- No system
- No actor
- Too many user interface details
  - “User types ID and password, clicks OK or hits Enter”
- Very low goal details
  - User provides name
  - User provides address
  - User provides telephone number.

# Writing Use Case Descriptions

1. Select a use case
  
2. Write abbreviated *full description* (Use case name, Scenario (if any), Actors, Flow of steps, Exception conditions)
  
3. For figuring Flow of steps,
  - Keep in mind general system model: Input-Processing-Output
  - Steps should be at nearly the same level of abstraction (each makes nearly same progress toward use case completion)
  
4. For figuring exception conditions, focus on if-then logic.

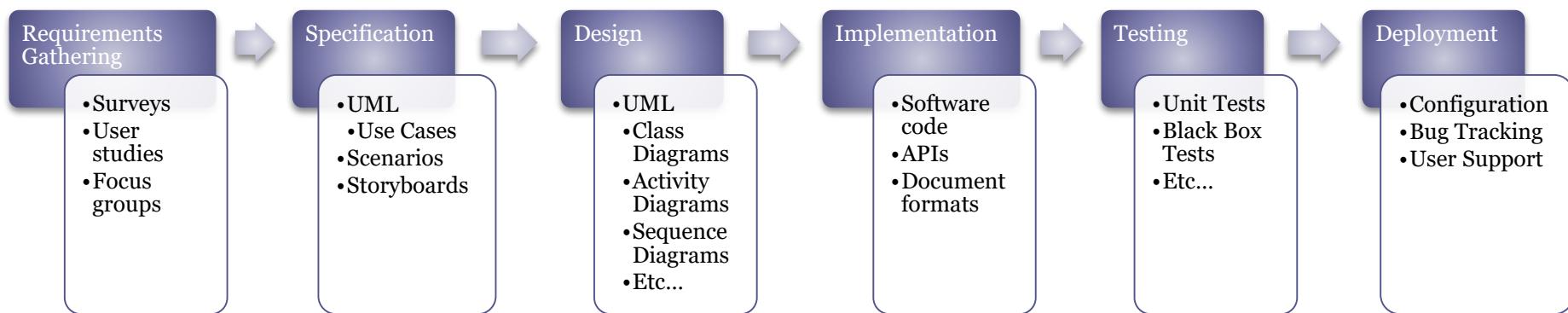
# Exercise

*Customer arrives at checkout with items to purchase in cash. Cashier records the items and takes cash payment. On completion, customer leaves with items.*

# Home Work Exercise

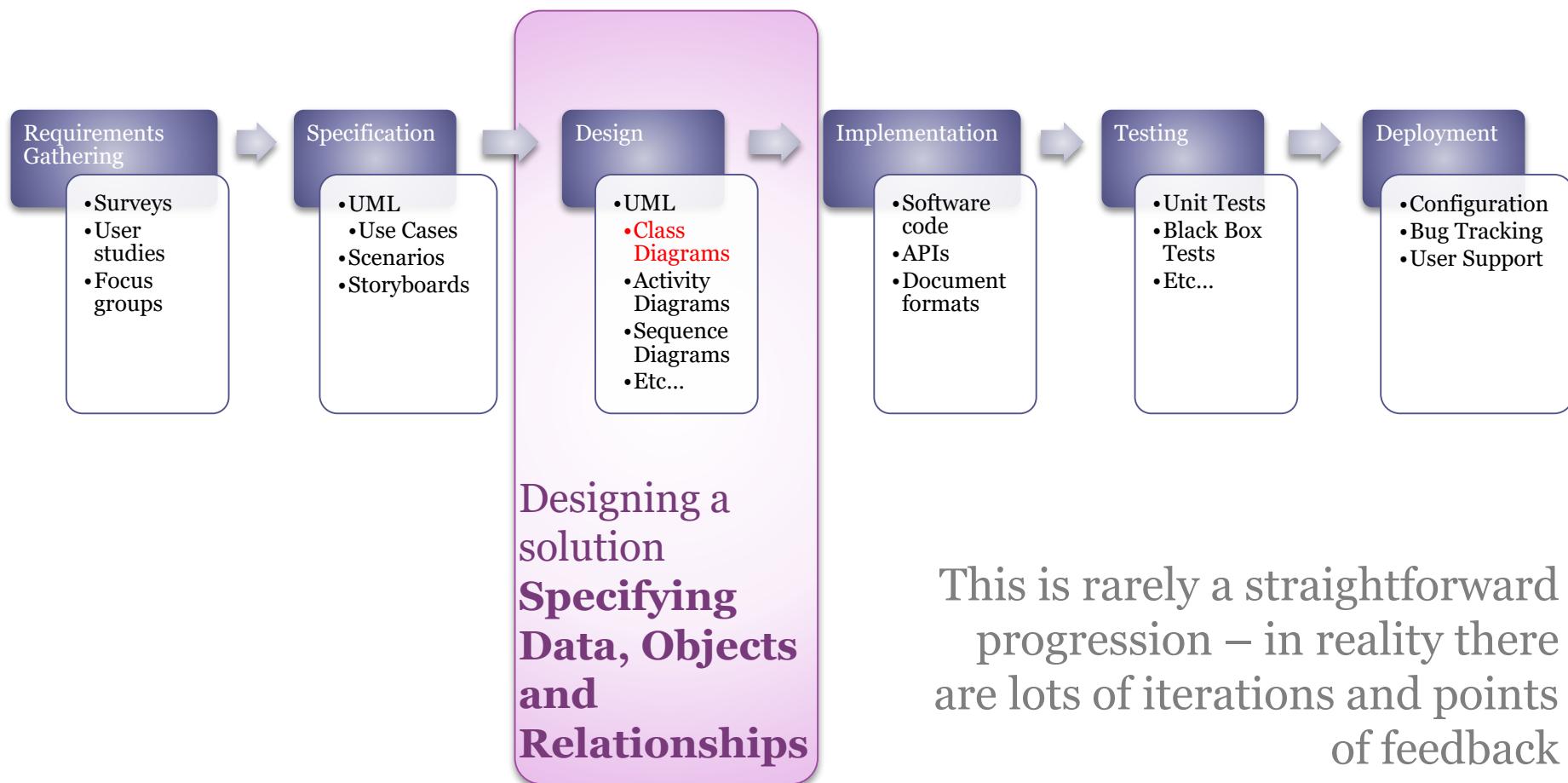
*A patient calls the clinic to make an appointment for a yearly checkup. The receptionist finds the nearest empty time slot in the appointment book and schedules the appointment for that time slot. "*

# OOAD: Big Picture



This is rarely a straightforward progression – in reality there are lots of iterations and points of feedback

# OOAD: Big Picture



# UML Class Diagrams

Lecture 6

# Types of Diagram

## Structure Diagrams

- Provide a way for representing the data and static relationships that are in an information system
- You are connecting different parts together to get the final design

## Behavioral Model

- Behavioral modeling refers to a way to model the system based on its functionality.

# Two Types of Diagram

## Behavior

### Interaction

Sequence

Communication

Interaction  
Overview

Timing

Activity

Use Case

State Transition

## Structure

Class

Object

Component

Composite

Deployment

Package

Profile

# What is UML Class Diagrams

- What is a UML class diagram? Imagine you were given the task of drawing a family tree. The steps you would take would be:
  - Identify the main members of the family
  - Identify how they are related to each other
  - Find the characteristics of each family member
  - Determine relations among family members
  - Decide the inheritance of personal traits and characters

# Basics of UML Class Diagrams

- A software application is comprised of classes and a diagram depicting the relationship between each of these classes would be the class diagram.
- A class diagram is a pictorial representation of the detailed system design

# Relationship between Class Diagram and Use Cases

- How does a class diagram relate to the use case diagrams that we learned before?

# Relationship between Class Diagram and Use Cases

- When you designed the **use cases**, you must have realized that the use cases talk about "**what are the requirements**" of a system.
- The aim of designing **classes** is to **convert this "what"** to a "**how**" for each requirement
- **Each use case** is further analyzed and broken up that form the basis for the classes that need to be designed

# Elements of a Class Diagram

- A class diagram is composed primarily of the following elements that represent the system's business entities:
  - **Class:** A class represents an entity of a given system. It provides captured implementation in the form of certain functionality of a given entity. These are exposed by the class to other classes as *methods*
  - Apart from functionality, a class also has properties that reflect unique features of a class. The properties of a class are called *attributes*.

# Naming Convention

Class naming: Use **singular names**

- Because each class represents a generalized version of a singular object.

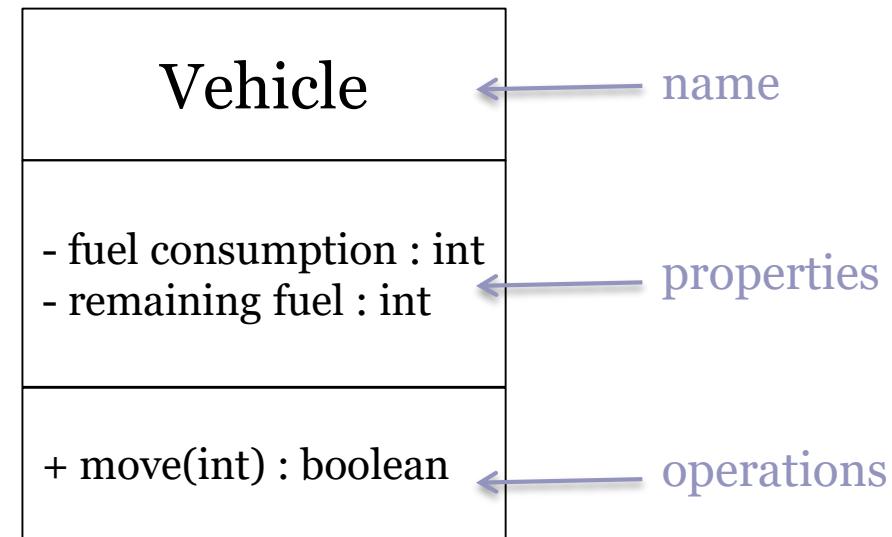
# Classes

We need to store several sorts of data about vehicles, including their fuel consumption and level of remaining fuel. Vehicles can move a given distance assuming that they have enough fuel.

- What is the name of the class?
- What are its properties?
- What are its operations?

# Classes

We need to store several sorts of data about vehicles, including their fuel consumption and level of remaining fuel. Vehicles can move a given distance assuming that they have enough fuel.



# Class Attribute

Person

```
+ name    : String  
# address : Address  
# birthdate : Date  
/ age      : Date  
- ssn      : Id
```

attributeName : Type

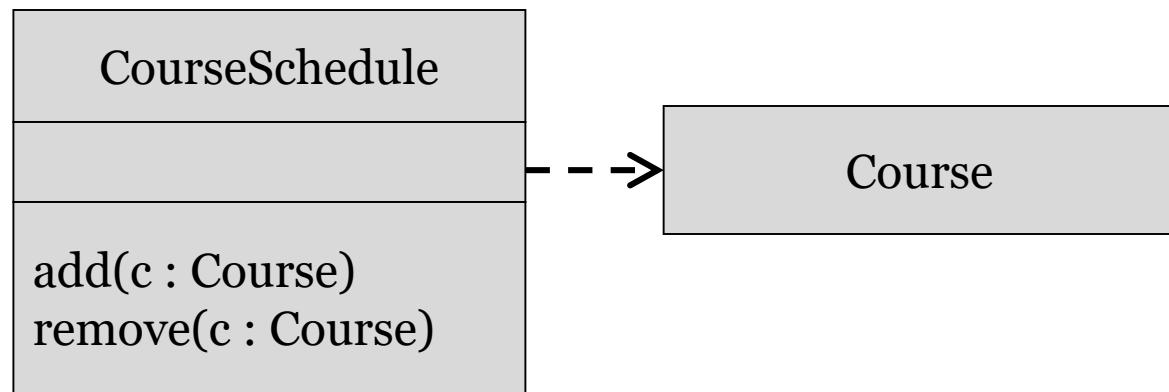
**“-” private**  
**“#” protected**  
**“+” public**  
**“/” derived**

# Relationships

- In UML, object interconnections (logical or physical), are modeled as relationships.
- There are three kinds of relationships in UML:
  - Dependencies
  - Generalizations
  - Associations

# Dependency

- Dependency is represented when a reference to one class is passed in as a method parameter to another class.



```
public class A {
```

```
    public void doSomething(B b) {
```

# Generalization

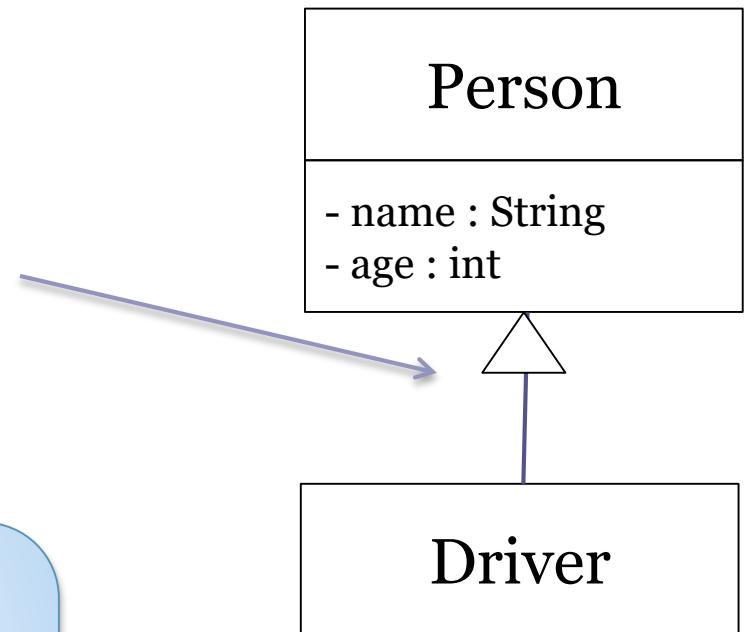
Drivers are a type of person. Every person has a name and an age.

# UML Class Diagrams: Generalization

Drivers are a type of person. Every person has a name and an age.

```
public Person {  
...  
} // class Person  
public class Driver extends Person{  
...  
} // class Driver
```

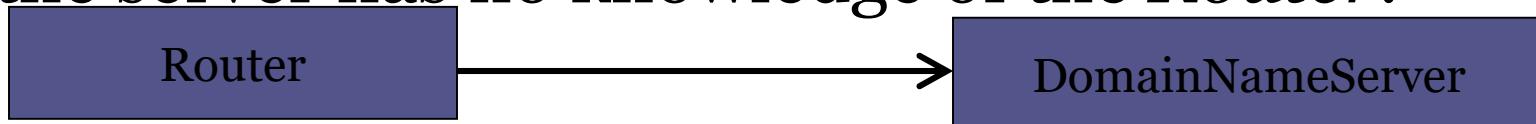
Note: we use a special kind of arrowhead to represent generalization



We assume that Driver **inherits** all the properties and operations of a Person (as well as defining its own)

# One-way Association

- We can constrain the association relationship by defining the *navigability* of the association.
- In one way association, We can navigate along a single direction only
- Denoted by an arrow towards the server object
- A *Router* object requests services from a *DNS* object by sending messages to (invoking the operations of) the server.
- The direction of the association indicates that the server has no knowledge of the *Router*.

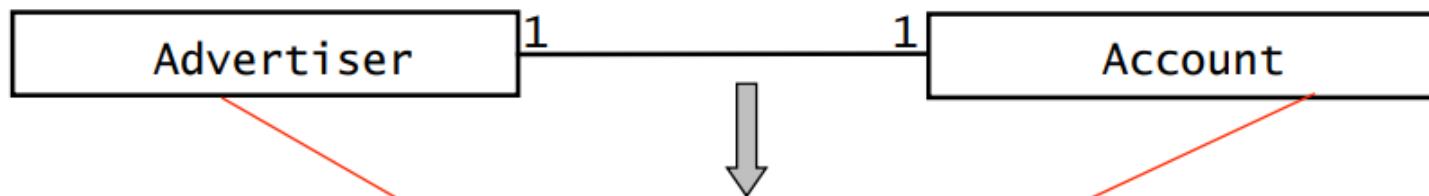


# One way Association-Person-Address

```
class Person {  
    string Name;  
    Address addr;  
    int Age;  
    public:  
        Person(){..}  
        ~Person{..}  
};
```

```
class Address {  
  
    string Street;  
    long postalCode;  
    string Area;  
    ....  
}
```

# One way Association



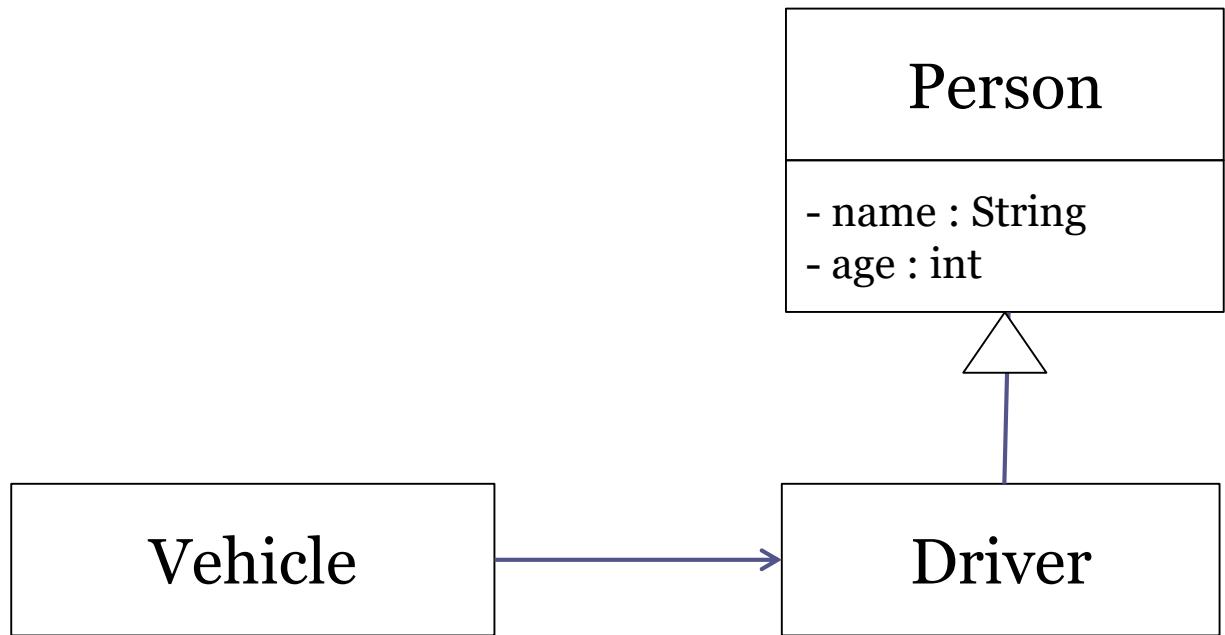
Source code after transformation:

```
public class Advertiser {  
    private Account account;  
    public Advertiser() {  
        account = new Account();  
    }  
    public Account getAccount() {  
        return account;  
    }  
}
```

One to one Relationship

# Composition

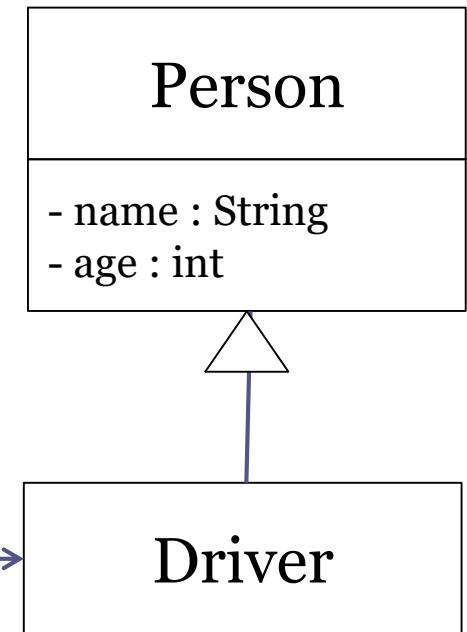
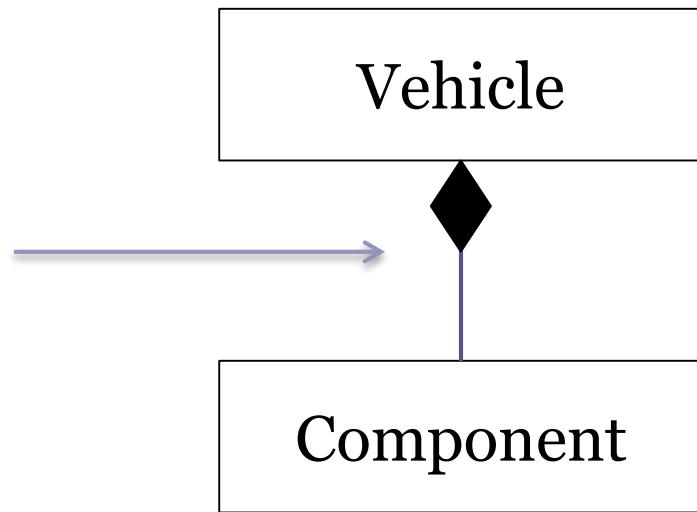
Vehicles are made up of many components.



# Composition

Vehicles are made up of many components.

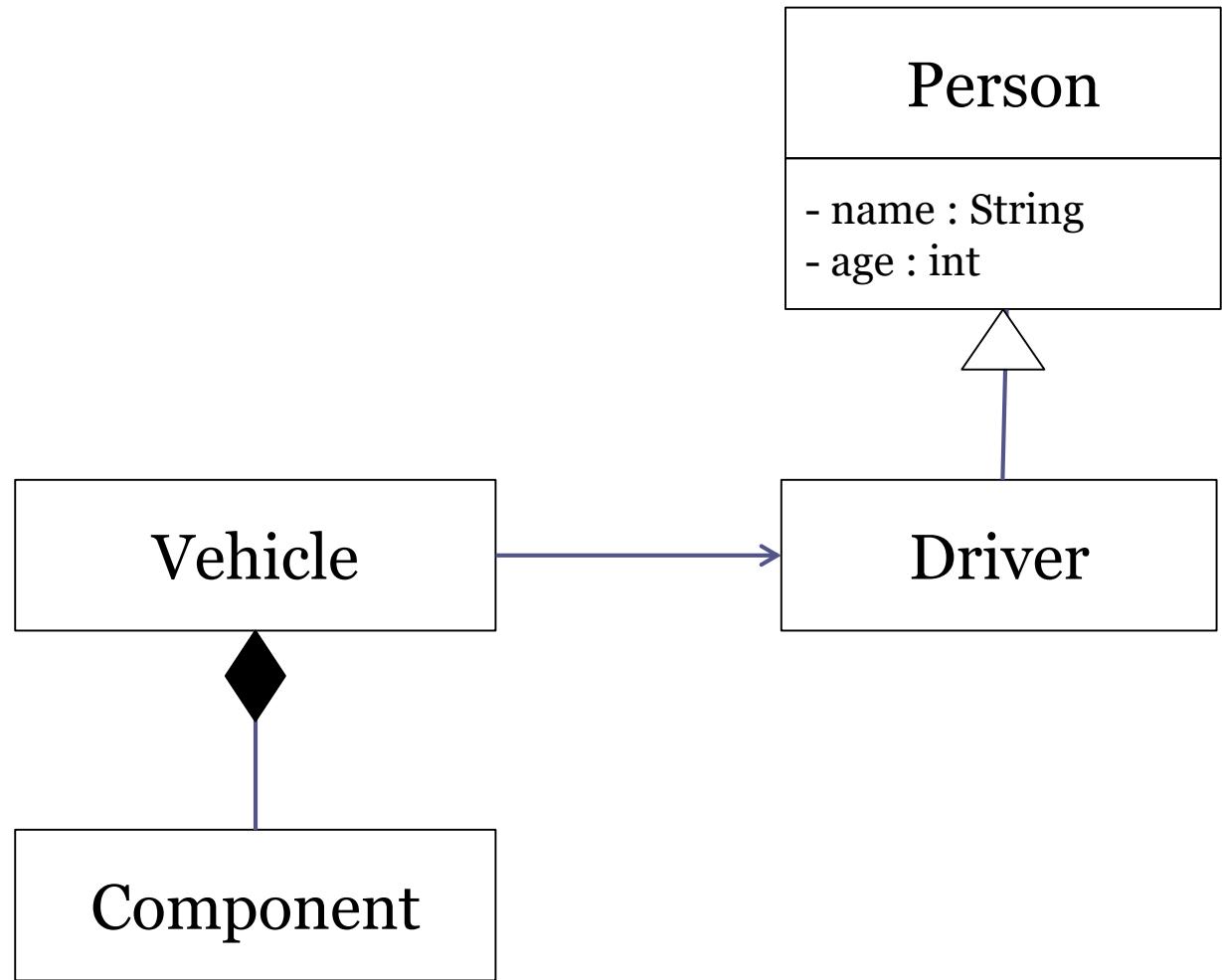
Note: we use a solid diamond to represent composition



We can use the composition relationship when there is a *strong lifecycle dependency* (i.e. a thing is only a component when it is part of a vehicle)

# Aggregation

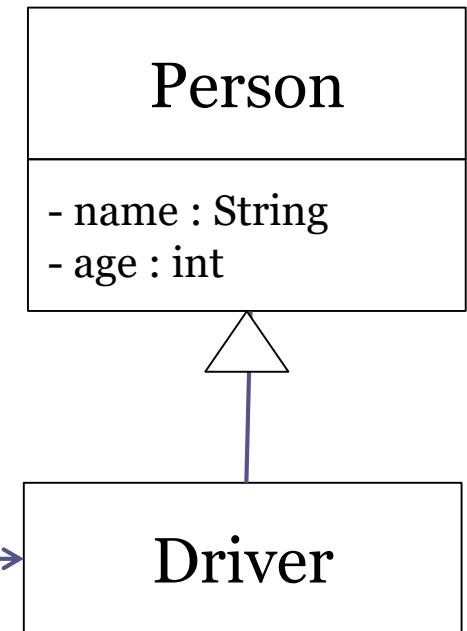
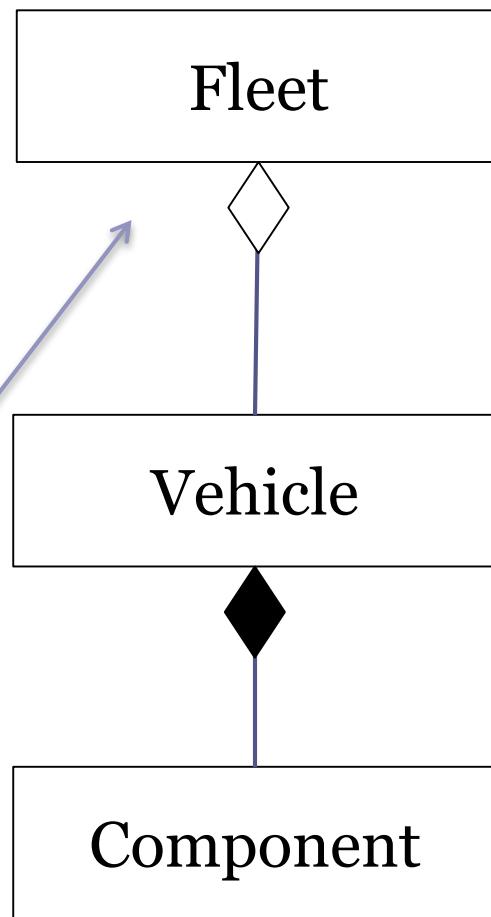
Vehicles are managed in a collection called a Fleet.



# Aggregation

Vehicles are managed in a collection called a Fleet.

Note: we use an empty diamond to represent aggregation



Use aggregation when there is a weak lifecycle dependency, i.e. vehicles can exist outside of a fleet

# Two Way Associations

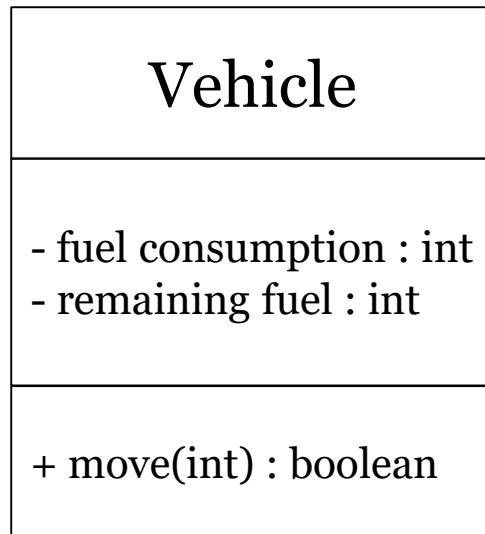
Vehicles  
always have  
at least one  
driver. Each  
driver must  
have a single  
vehicle.

Vehicle
- fuel consumption : int - remaining fuel : int
+ move(int) : boolean

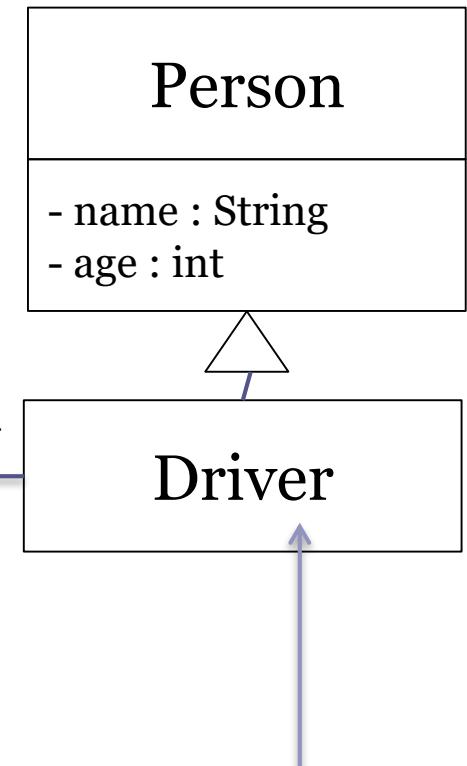
# Two way Associations

Vehicles always have at least one driver. Each driver must have a single vehicle.

Associations are drawn as solid lines. *Multiplicity* is written at the target ends



Associations can optionally have a name and visibility



Note that properties and operations are optional

# Two-way Association(Bidirectional)

- We can navigate in both directions
- Denoted by a line between the associated objects

Employee  $\frac{\text{works-}}{* \text{ for } 1}$  Company

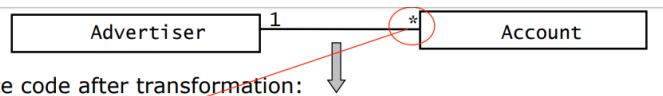
- Employee works for company
- Company employs employees

# Two way Association-Contractor-Project

```
class Contractor
{
private:
string Name;
Project MyProject;
...
};
```

```
class Project
{
string Name;
Contractor person;
.....
};
```

# Bidirectional Association

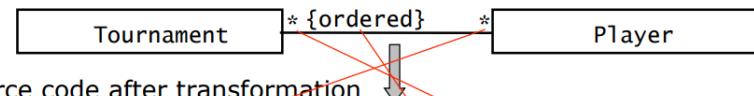


## ~~Source code after transformation:~~

```
public class Advertiser {
    private Set accounts;
    public Advertiser() {
        accounts = new HashSet();
    }
    public void addAccount(Account a) {
        accounts.add(a);
        a.setOwner(this);
    }
    public void removeAccount(Account a)
    {
        accounts.remove(a);
        a.setOwner(null);
    }
}
```

```
public class Account {  
    private Advertiser owner;  
    public void setOwner(Advertiser  
newOwner) {  
        if (owner != newOwner) {  
            Advertiser old = owner;  
            owner = newOwner;  
            if (newOwner != null)  
                newOwner.addAccount(this);  
            if (oldOwner != null)  
                old.removeAccount(this);  
        }  
    }  
}
```

## One to many



### ~~Source code after transformation~~

```
public class Tournament {  
    private List players;  
    public Tournament() {  
        players = new ArrayList();  
    }  
    public void addPlayer(Player p)  
    {  
        if (!players.contains(p)) {  
            players.add(p);  
            p.addTournament(this)  
        }  
    }  
}
```

```
public class Player {
    private List tournaments;
    public Player() {
        tournaments = new
ArrayList();
    }
    public void
addTournament(Tournament t) {
    if
(!tournaments.contains(t)) {
        tournaments.add(t);
        t.addPlayer(this);
    }
}
```

## many to many



~~Source code after transformation:~~

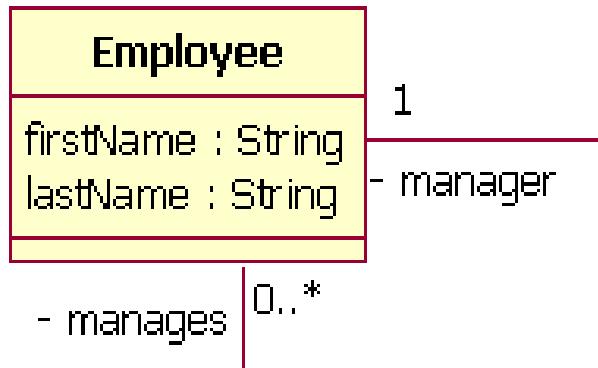
```
public class Advertiser {  
    /* account is initialized  
     * in the constructor and never  
     * modified. */  
    private Account account;  
    public Advertiser() {  
        account = new  
        Account(this);  
    }  
    public Account getAccount() {  
        return account;  
    }  
}
```

```
public class Account {
    /* owner is initialized
     * in the constructor and
     * never modified. */
    private Advertiser owner;
    public Account(owner:Advertiser)
        this.owner = owner;
    }
    public Advertiser getOwner() {
        return owner;
    }
}
```

## One to one

# Self Association

A class can have *a self association/ reflexive Association.*



Two instances of the same class:  
Pilot  
Aviation engineer

# Self Association

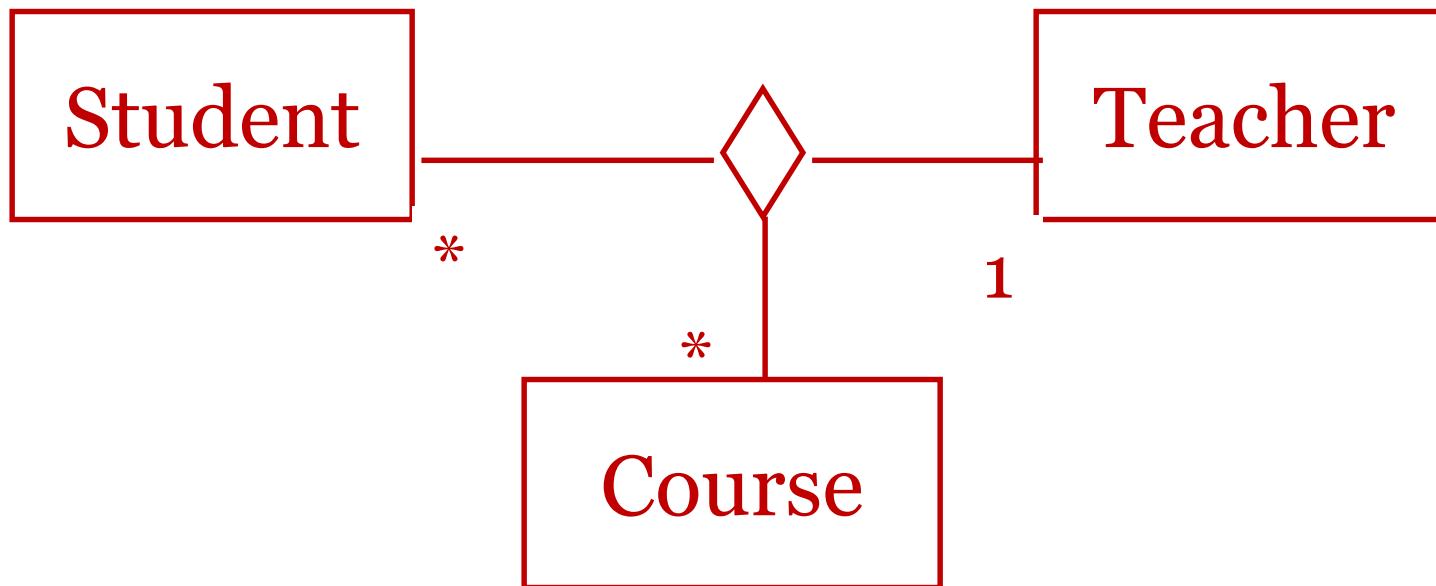
```
class Course
{
private:
    std::string m_name;
    Course *m_prerequisite;

public:
    Course(std::string &name, Course *prerequisite=nullptr):
        m_name(name), m_prerequisite(prerequisite)
    {
    }

};
```

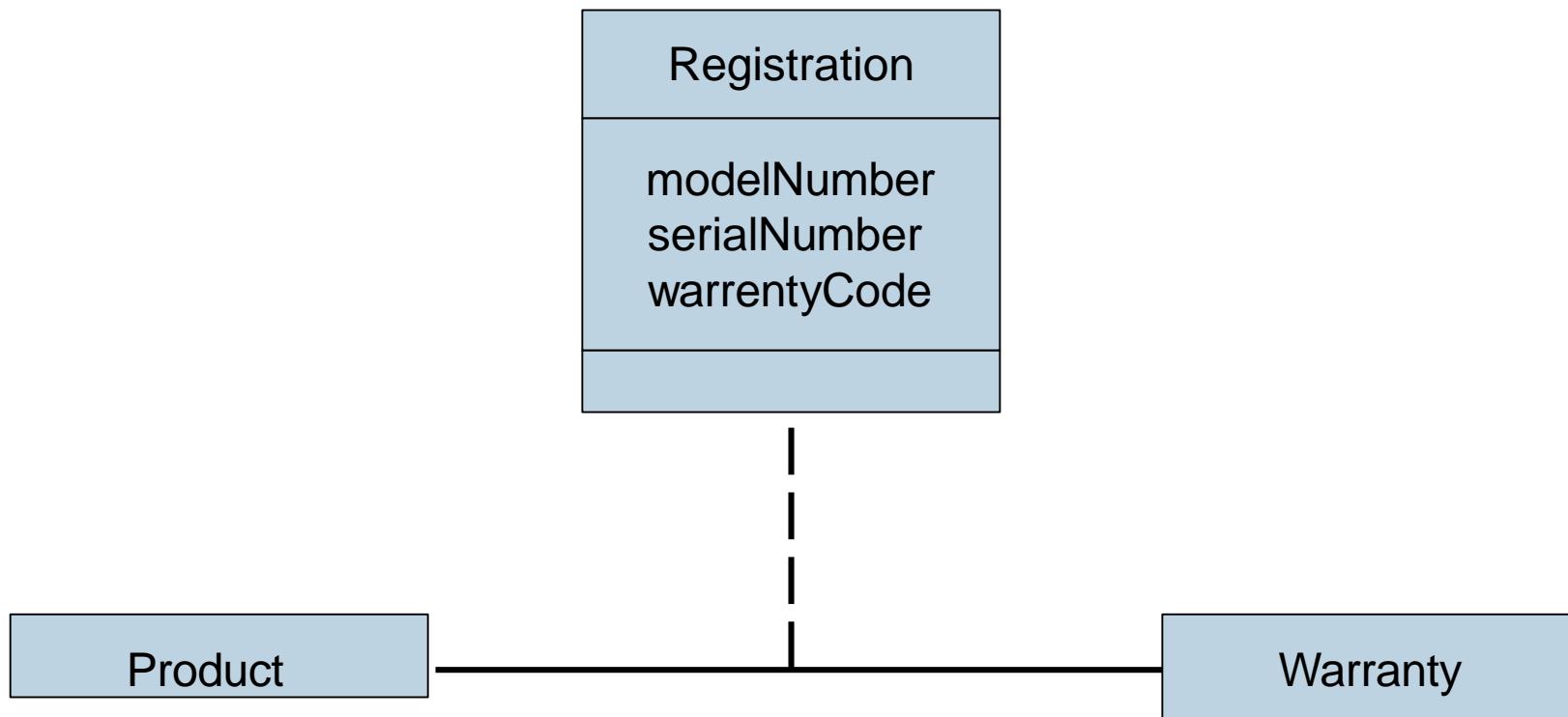
# N-Ary Association

- Associates objects of more than two classes.
- Denoted by a diamond with lines connected to associated objects.



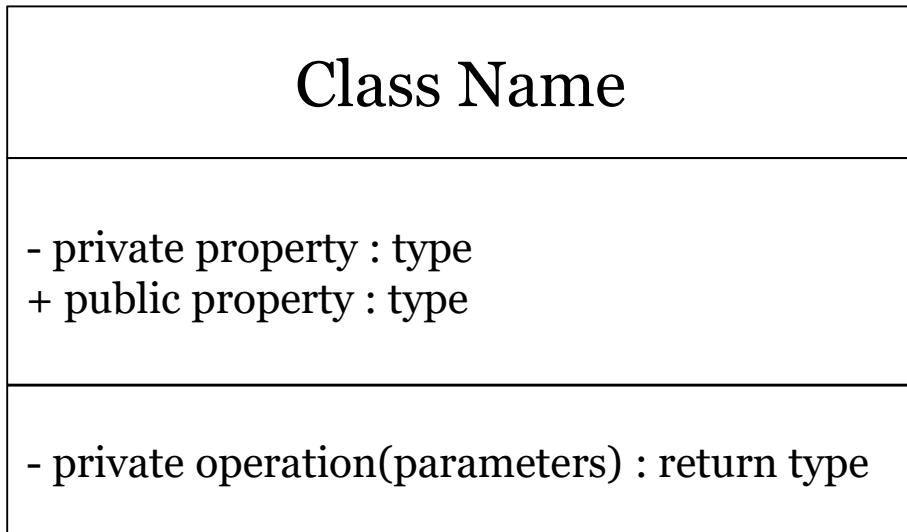
# Association Class

- Associations can also be objects themselves, called link classes or an association classes.
- A link is an instance of an association.



# UML Class Diagrams

Class



Association



Generalization (Inheritance)



Composition (strong)

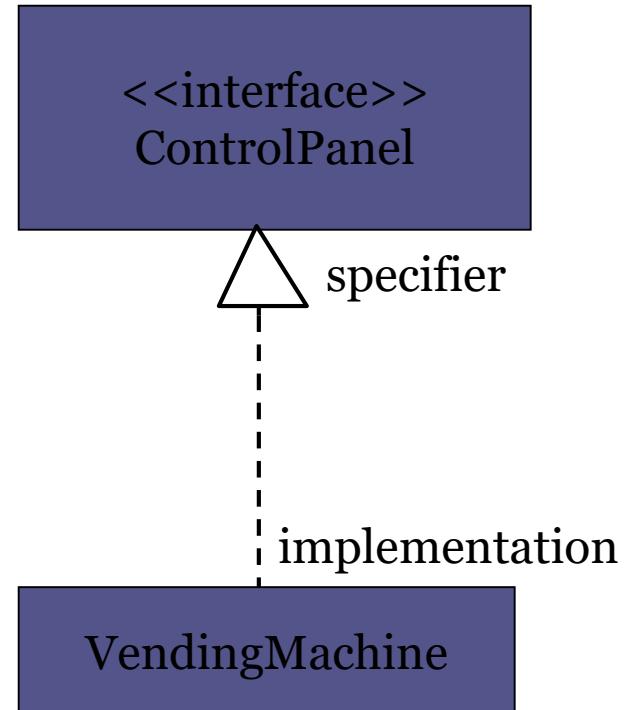


Aggregation (weak)



# Interface Realization Relationship

A *realization* relationship connects a class with an interface that supplies its behavioral specification. It is rendered by a dashed line with a hollow triangle towards the specifier.

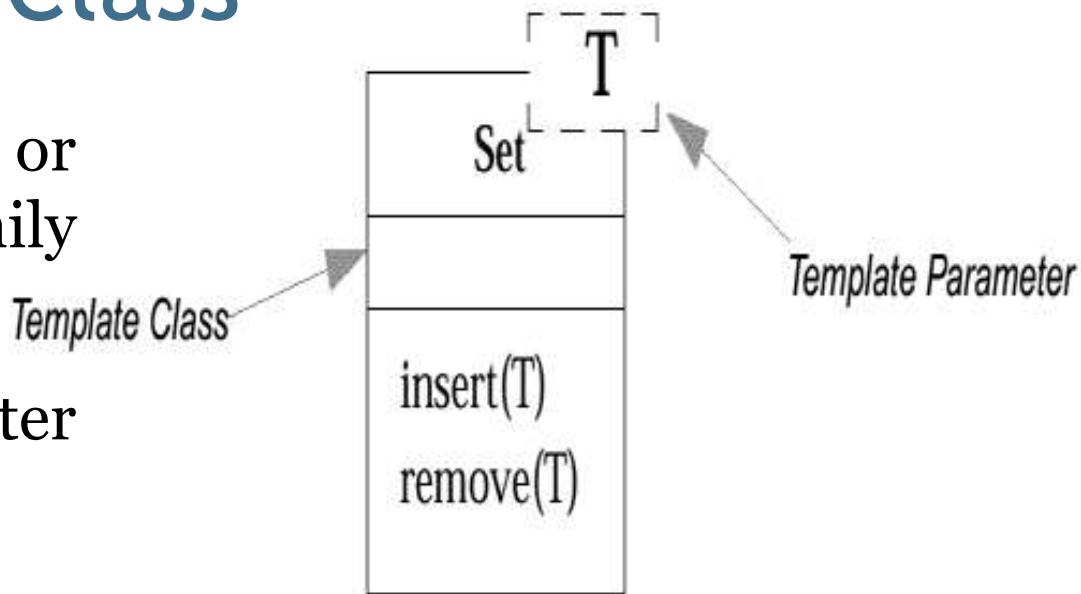


```
public interface A {  
}  
} // interface A  
  
public class B implements A {  
}  
} // class B
```

# Parameterized Class

A *parameterized class* or *template* defines a family of potential elements.

To use it, the parameter must be bound.



```
class Set <T> {  
    void insert (T newElement);  
    void remove (T anElement);
```

# Enumeration

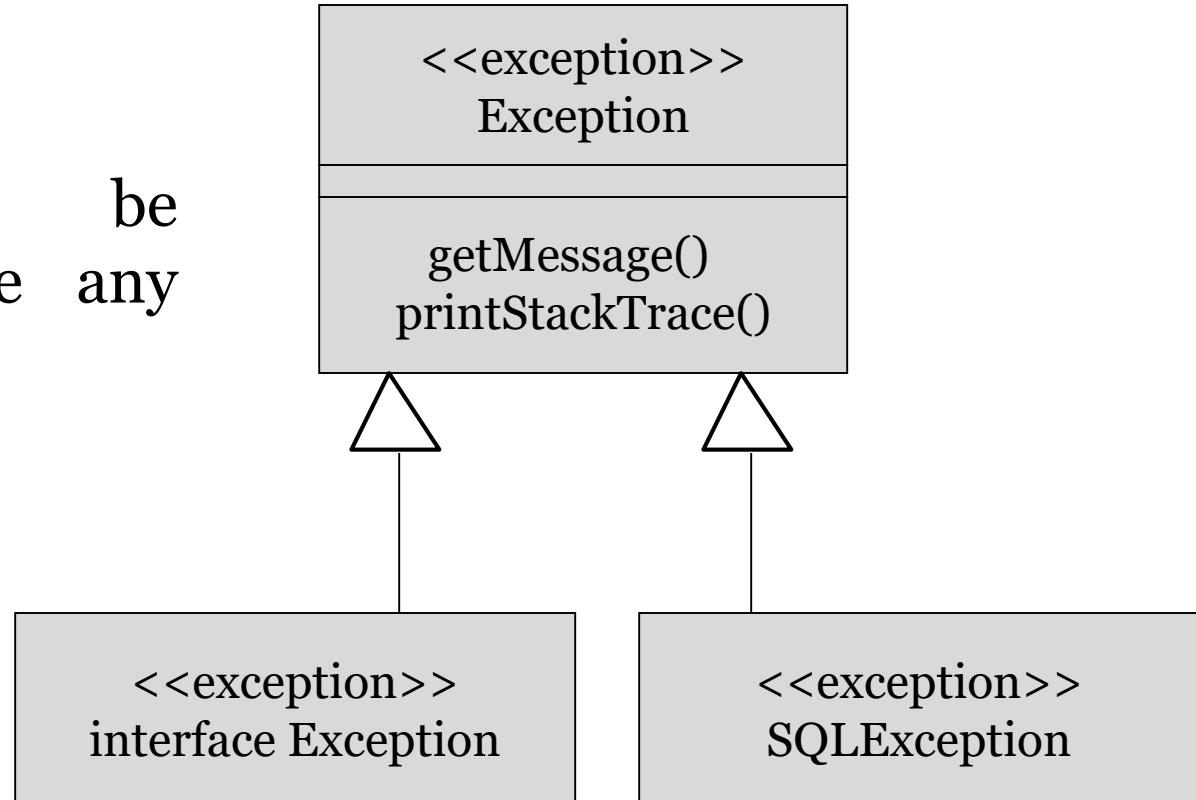
An *enumeration* is a user-defined data type that consists of a name and an ordered list of enumeration literals.

<<enumeration>>  
Boolean

false  
true

# Exceptions

*Exceptions* can be modeled just like any other class.



# Package

provides the ability to group together classes and/or interfaces that are either similar in nature or related.

Grouping these design elements in a package element provides for better readability of class diagrams, especially complex class diagrams.

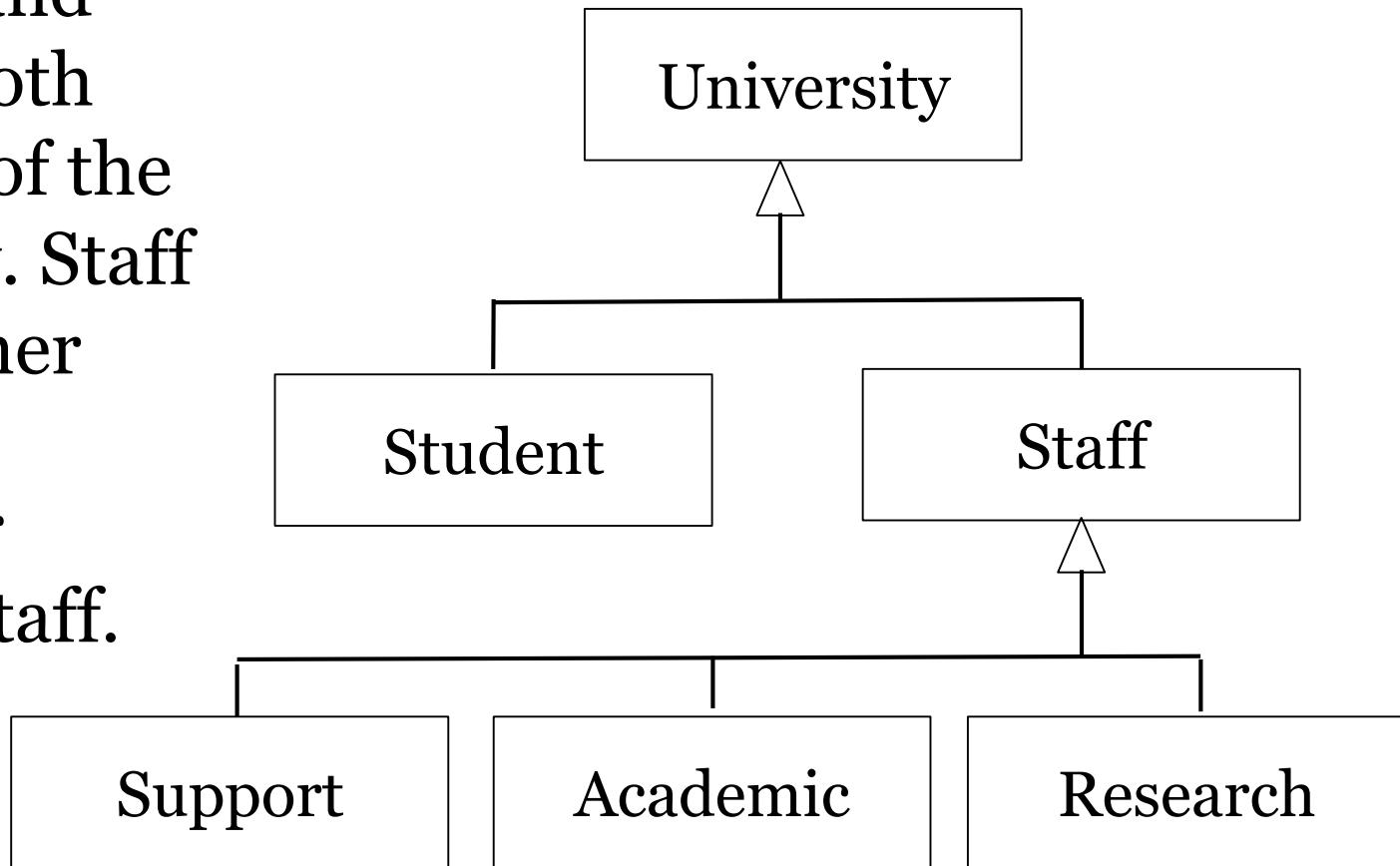
com.novusware.cms.bo

# How might we model...

Students and staff are both members of the University. Staff can be either academic, support or research staff.

# How might we model...

Students and staff are both members of the University. Staff can be either academic, support or research staff.

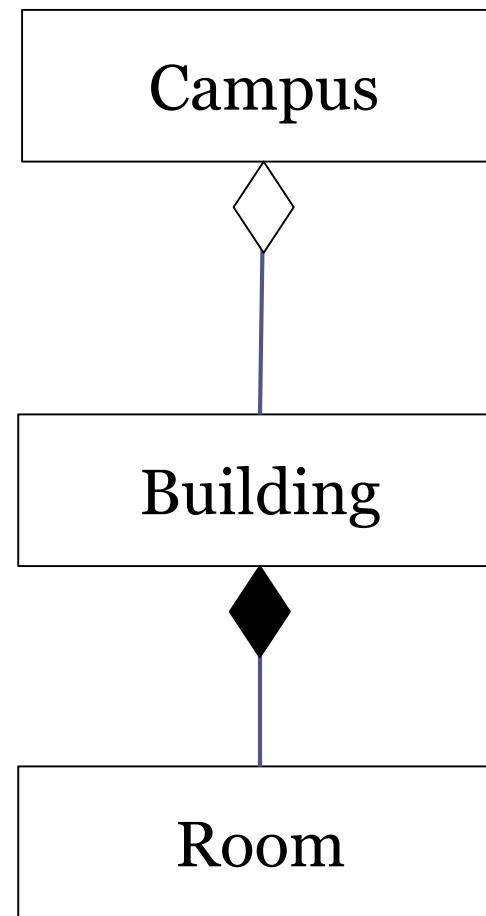


# How might we model...

A campus is  
made up of  
many  
buildings. A  
building is  
made of many  
rooms.

# How might we model...

A campus is made up of many buildings. A building is made of many rooms.

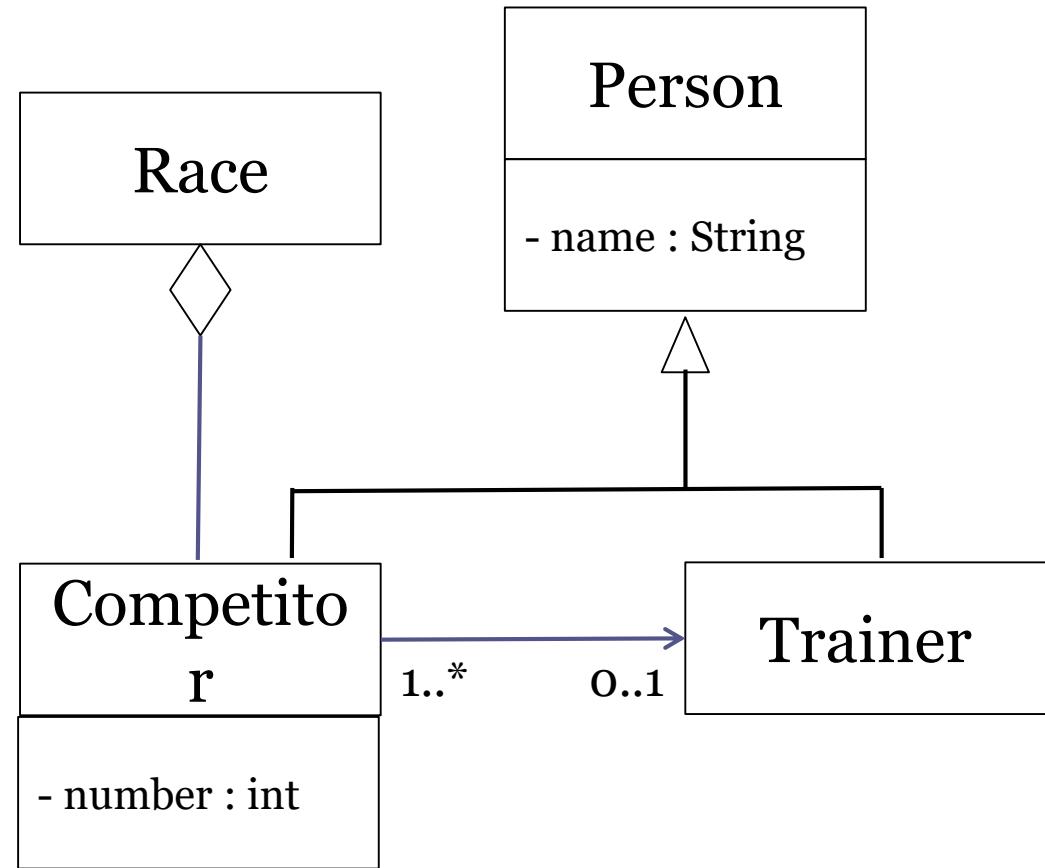


# How might we model...

A Race has many competitors. Each competitor may have a trainer. Both types of person have a name, but competitors also have a number.

# How might we model...

A Race has many competitors. Each competitor may have a trainer. Both types of person have a name, but competitors also have a number.



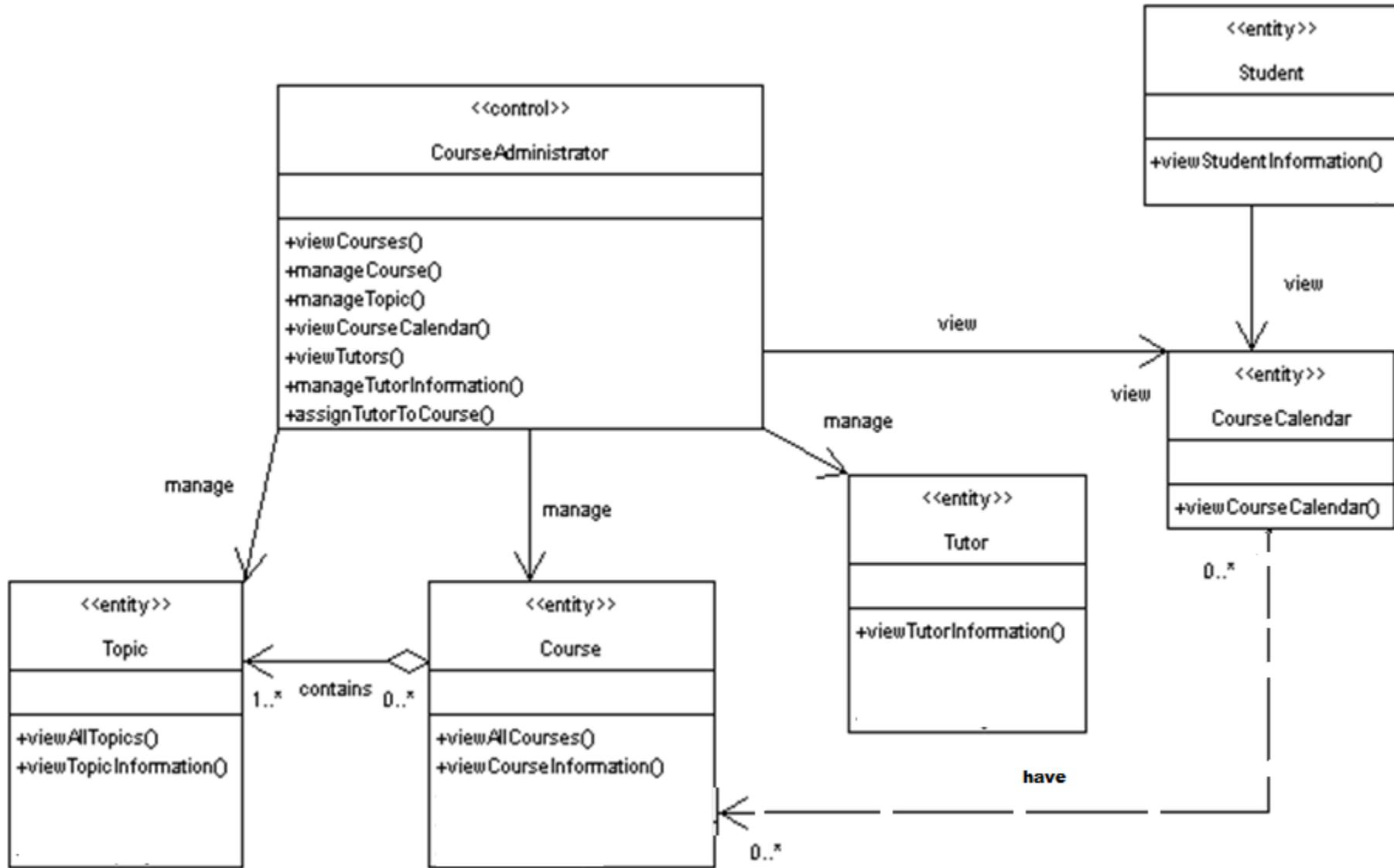
# The UML Class Diagram in Action

## Case study—Courseware Management System

- In the **use case lecture**, we **identified the primary actors and use cases** in the use case model of the case study.
- Let us **recap the analysis** that was performed when the use case model was designed.
- The **following terms and entities** specific to the system **were identified** from the problem statement:
  - Courses and **Topics** that make up a course
  - **Tutors** who teach courses
  - Course administrators who mange the assignment of the courses to tutors
  - **Calendar** or Course Schedule is generated as a result of the Students who refer to the Course schedule or Calendar to decide which courses for which they wish to sign up
  - **Student** can view his information

# The UML Class Diagram in Action....

Identifying relationship and drawing the class diagram



# Home Work Exercises



# University Team Management

- In the OOAD course at Fast University, students are member of teams.
- Each team has 2 or 3 members.
- Each team completes 0 to 3 assignments.
- Each student takes exactly two midterm test.
- Computer Science students have a single account on Coding Development facility , while each engineering student has an account on the Engineering facility.
- Each assignment and midterm is assigned a mark.

# University System

- A Fast university offers degrees to students.
- The university consists of faculties each of which consists of one or more departments.
- Each degree is administered by a single department.
- Each student is studying towards a single degree.
- Each degree requires one to 20 courses.
- A student enrolls in 1-5 courses (per term).
- A course cab be either graduate or undergraduate, but not both.
- Likewise, students are graduates or undergraduates but not both.

# Library System

- *This application will support the operations of a technical library for an R&D organization. This includes the searching for and lending of technical library materials, including books, videos, and technical journals. Users will enter their company ids in order to use the system; and they will enter material ID numbers when checking out and returning items. Each borrower can be lent up to five items. Each type of library item can be lent for a different period of time (books 4 weeks, journals 2 weeks, videos 1 week). If returned after their due date, the library user's organization will be charged a fine, based on the type of item( books \$1/day, journals \$3/day, videos \$5/day).Materials will be lent to employees with no overdue lendable, fewer than five articles out, and total fines less than \$100.*

Draw the UML class diagram showing the domain model for online shopping. The purpose of the diagram is to introduce some common terms, "dictionary" for online shopping - Customer, Web User, Account, Shopping Cart, Product, Order, Payment, etc. and relationships between. It could be used as a common ground between business analysts and software developers.

- Each customer has unique id and is linked to exactly one account. Account owns shopping cart and orders. Customer could register as a web user to be able to buy items online. Customer is not required to be a web user because purchases could also be made by phone or by ordering from catalogues. Web user has login name which also serves as unique id. Web user could be in several states - new, active, temporary blocked, or banned, and be linked to a shopping cart. Shopping cart belongs to account.
- Account owns customer orders. Customer may have no orders. Customer orders are sorted and unique. Each order could refer to several payments, possibly none. Every payment has unique id and is related to exactly one account.
- Each order has current order status (new, hold, shipped, delivered, closed). Both order and shopping cart have line items linked to a specific product. Each line item is related to exactly one product. A product could be associated to many line items or no item at all.

# Car Rental System

We want to develop a car rental system. When any customer/user wants to rent a car, he/she must be registered with the system. The system must also create the list of cars according to their category (i.e. Gomini, Go, Goplus). When a user requests booking of a car of certain category, start day of rental is the current day or a day after the current day; end day of rental lies after the start day. Customer can also cancel his or her booking. When the customer requests a pickup, a suitable car must be found among the currently available cars. When the trip is completed, booking closes and car must be returned.

# Banking System Application

- We have to develop a banking system application which provides many services to the customers like opening and closing accounts, balance enquiry, deposit money, cash withdrawal, and taking cards. Customer can open two types of accounts i.e. saving and current account. Bank also has an ATM machine which provides the services related to balance. Customer can take loan from the bank against his/her account. One customer can take only one loan at a time.

# Organization Scenario

- Consider the world of companies:
- Companies employ employees (who can only work for one company), and consist of one or more departments.
- Each company has a single president, who is an employee.
- Departments have employees as members and run projects (one or more.)
- Employees can work in 1 to 3 projects, while a project can have 2 to 50 assigned employees.
- You may assume that companies have a name and address, while employees have a emp# and a salary



# Entity, Control & Boundary Classes

## Lecture 7



# Class Diagram

- Class diagram describes the types of objects in the system and the various kinds of relationships that exist among them.
- It also shows the attributes and services of a class and the constraints that apply to the way objects are connected.

# Class

Stereotype:  
Type of the Class;  
**entity** for Analysis.

Name of the Class



<<entity>>

Patient

Attributes of the  
Class



-name : CHAR

-dateOfBirth: DATE

Operations of the  
Class



+getName ()



# Use Case Realization

- A use-case realization describes how a particular use case is realized within the Design Model, in terms of collaborating objects.
  - A use-case realization is **one possible realization** of a use case.
  - A use-case realization in the Design Model can be ***traced*** to a use case in the Use-Case Model.
- Used to remind us the connection between the requirements expressed as use case and the object design that satisfies the requirement.



# Diagramming Representation of Use Case Realization

- **Interaction Diagrams (sequence and/or collaboration diagrams)** can be used to describe how the use case is realized in terms of collaborating objects.
  - These diagrams model the detailed collaborations of the use-case realization.
- **Class Diagrams** can be used to describe the **classes** that participate in the realization of the use case, as well as their supporting relationships.
  - The use-case realization can be used by class designers to understand the class's role in the use case and how the class interacts with other classes.

## Use Case Model

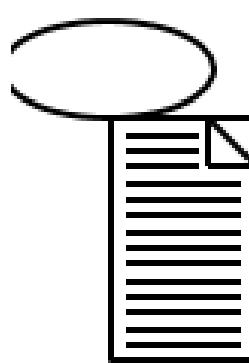


Use Case

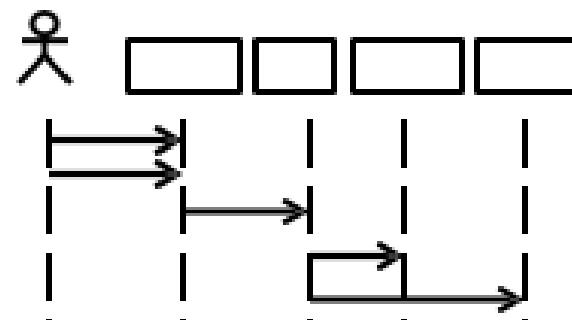
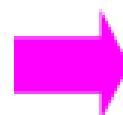
## Design Model



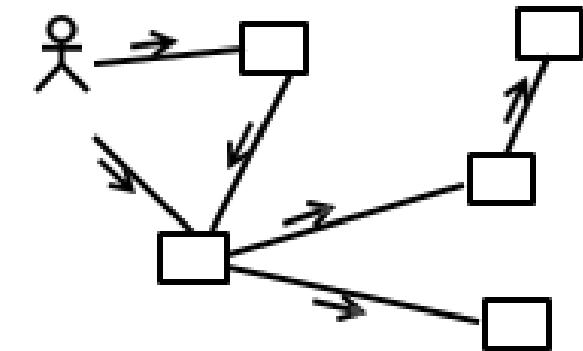
Use-Case Realization



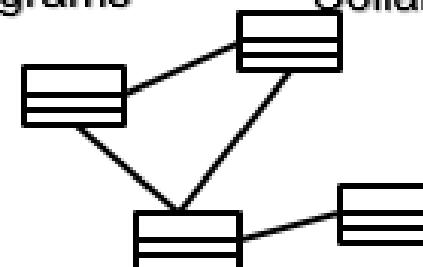
Use Case



Sequence Diagrams



Collaboration Diagrams



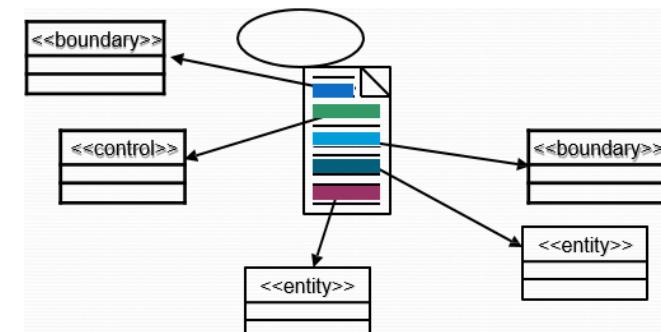
Class Diagrams

Not what we have so far.  
These are Design Classes.



# Identifying Candidate Classes from Behavior

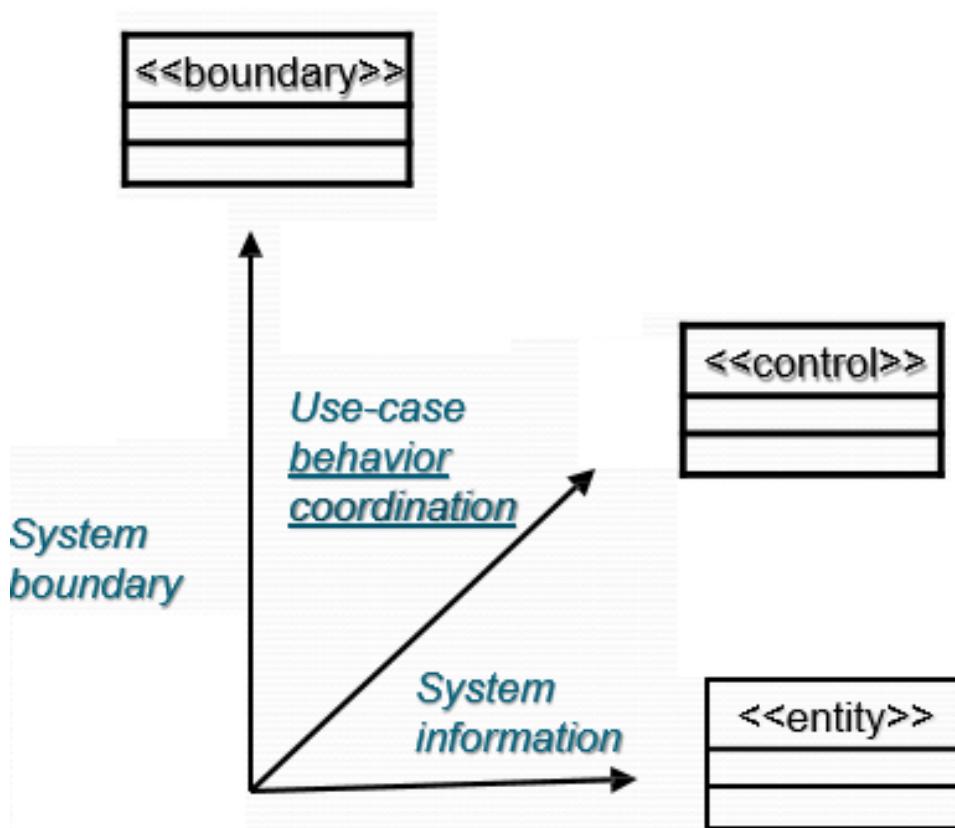
- Will use three perspectives of the system to identify these classes.
  - The '**boundary**' between the system and its actors
  - The '**entity**' the system uses
  - The '**control logic**' of the system
- Will use stereotypes to represent these perspectives (**boundary**, **control**, **entity**)
  - These are **conveniences** used during analysis that will disappear or be transitioned into different design elements during the design process.
- Will result in a **more robust model** because these are the three things that are most likely to change in system and so we **isolate** them so that we can treat them separately.



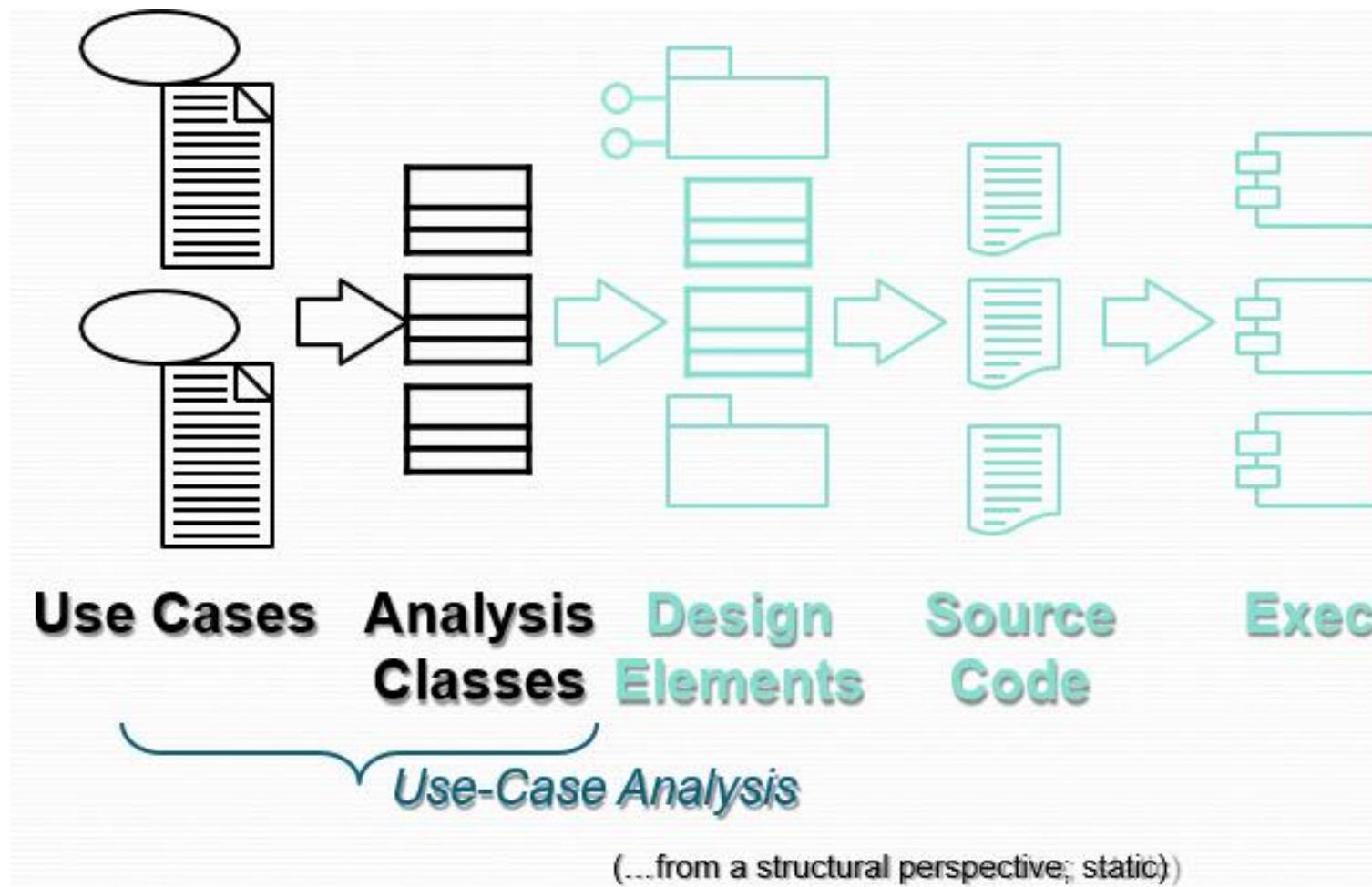
# What is an Analysis Class?

Can use with the name of the stereotype  
In guillemets or as symbols with unique icons.

Finding a candidate set of classes is the first part of transforming a mere statement of required behavior to a description of how the System will work.



# Analysis Classes: First Step towards Executables





# Application Class Stereotypes

- «boundary»
  - Provides the interface for external interactions
  - 1:1 correlation with «external» classes
- «entity»
  - Manages persistent data
- «control»
  - Central controller for one or more use case scenarios

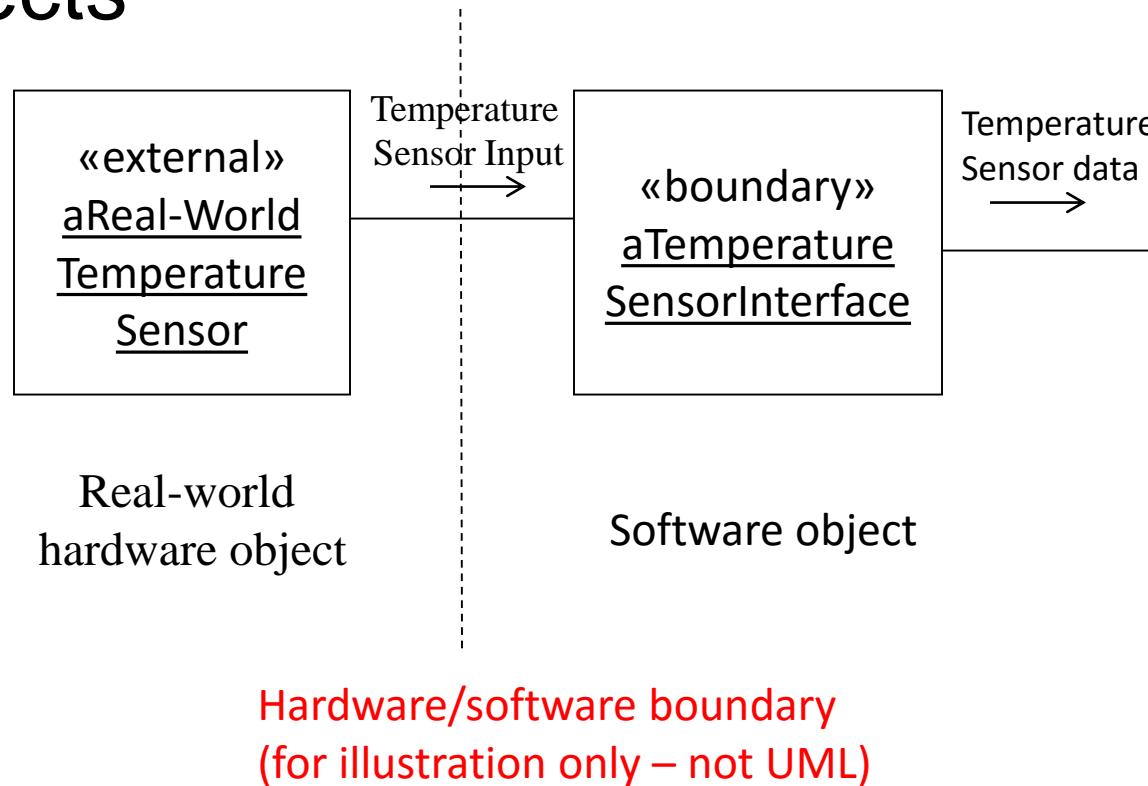


# Boundary Classes

- Provide the interface to a user or another system.
- Handles communication between system surroundings and the inside of the system
  - User interface classes
  - System interface classes
  - Device interface classes

# Identifying «boundary» Classes

- One-to-one mapping with «external» objects



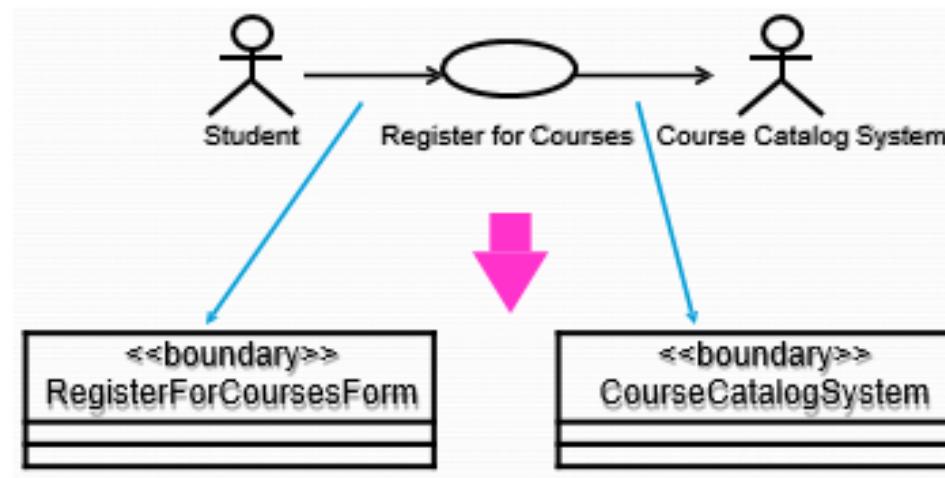


# Boundary Class

- For the initial identification of boundary classes is **one boundary class per actor/use-case pair**.
  - This class can be viewed as having responsibility for coordinating the interaction with the actor.
  - This may be refined as a more detailed analysis is performed.
  - This is particularly true for window-based GUI applications, where there is typically one boundary class for each window, or one for each dialog.

# Example: Finding Boundary Classes

- One boundary class per actor/use case pair:



- The RegisterForCoursesForm contains a Student's "schedule-in-progress". It displays a list of Course Offerings for the current semester from which the Student may select to be added to his/her Schedule.
- The CourseCatalogSystem interfaces with the legacy system that provides the complete catalog of all courses offered by the university.



# Boundary Class (Interface)

- User Interface Classes
  - Concentrate on what information is presented to the user
  - Do NOT concentrate on the UI details
  - Analysis Classes are meant to be a first cut at the abstraction of the system.
  - The boundary classes may be used as ‘holding places’ for GUI classes.
  - Do not do a GUI design in analysis, but isolate all environment-dependent behavior. (Likely you may be able to reverse engineer a GUI component and tailor it.)
  - The expansion, refinement and replacement of these boundary classes with actual user interface classes is a very important activity of Class Design.
  - If prototyping the interface has been done, these screen dumps or sketches may be associated with a boundary class.
  - Only model the key abstractions of the system – not every button, list, etc. in the GUI.



# Boundary Class (System & Device)

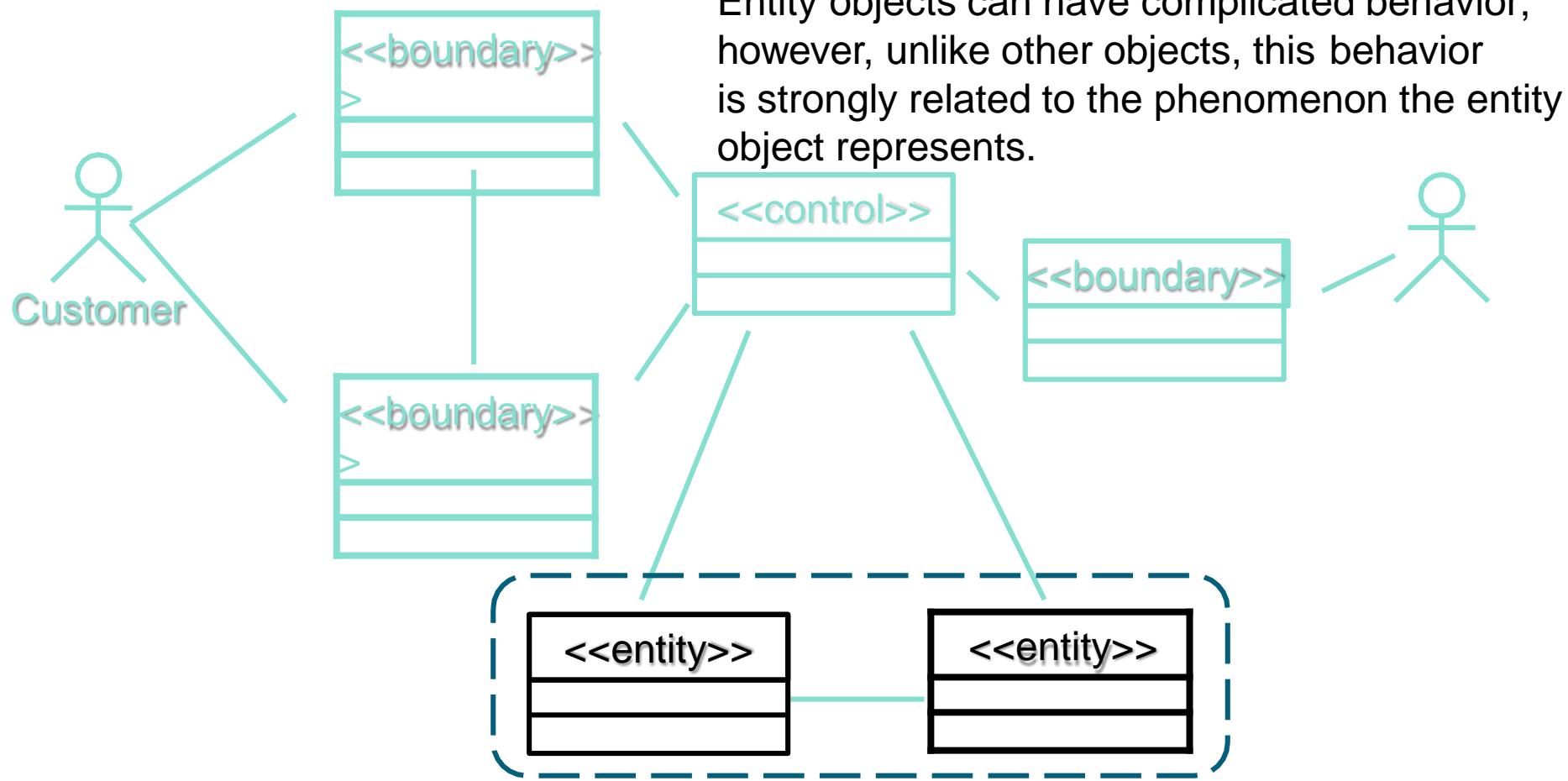
- System and Device Interface Classes
  - Concentrate on what protocols must be defined
    - Note that your application must interface with an existing 'information source.'
  - Do NOT concentrate on how the protocols will be implemented
- If the interface to an existing system or device is already well-defined, the boundary class responsibilities should be derived directly from the interface definition.
- If there is a working communication with the external system or device, make note of it for later reference during design.



# Entity Classes

- Fundamental building block which perform internal tasks
- Represent real world entity
- Entity classes represent stores of information in the system
- They are typically used to represent the key items the system manages.
- Entity objects (instances of entity classes) are used to hold and update information about some phenomenon, such as an event, a person, or some real-life object. (advisor, teacher, university, student etc.)
- They are usually persistent, having attributes and relationships needed for a long period, sometimes for the life of the system.
- The main responsibilities of entity classes are to store and manage information in the system.

# The Role of an Entity Class



**Store and manage information in the system**

The values of its attributes and relationships are often given by an actor.  
Entity objects are independent of the environment (actors)



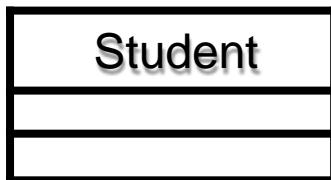
# Candidate Entity Classes

- Sometimes there is a need to model information about an actor within the system. This is not the same as modeling the actor (actors are external by definition).
- For example, a course registration system maintains information about the student which is independent of the fact that the student also plays a role as an actor of the system.
- This information about the student that is stored in a ‘Student’ class is completely independent of the ‘actor’ role the student plays; the Student class (entity) will exist whether or not the student is an actor to the system.

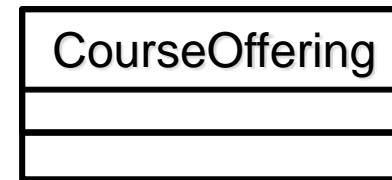


# Example: Candidate Entity Classes

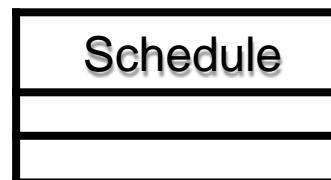
Register for Courses (Create Schedule)



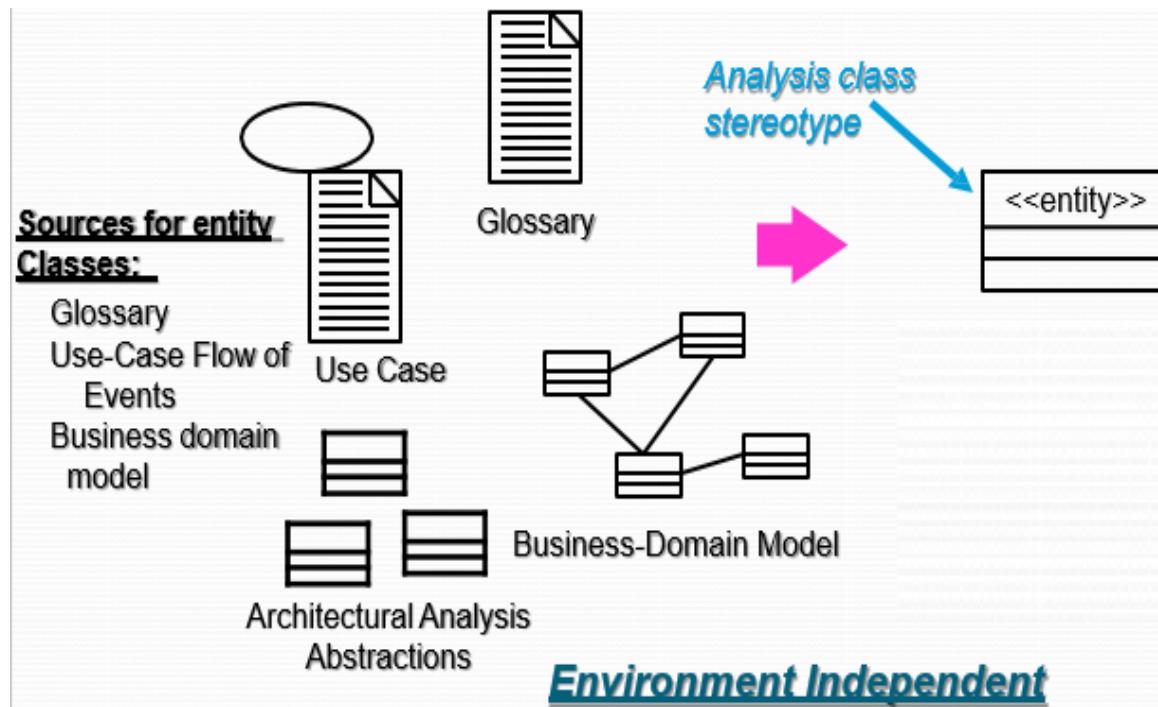
A person enrolled in classes at the university



A specific offering for a course including days of week and times



The courses a student has selected for current semester

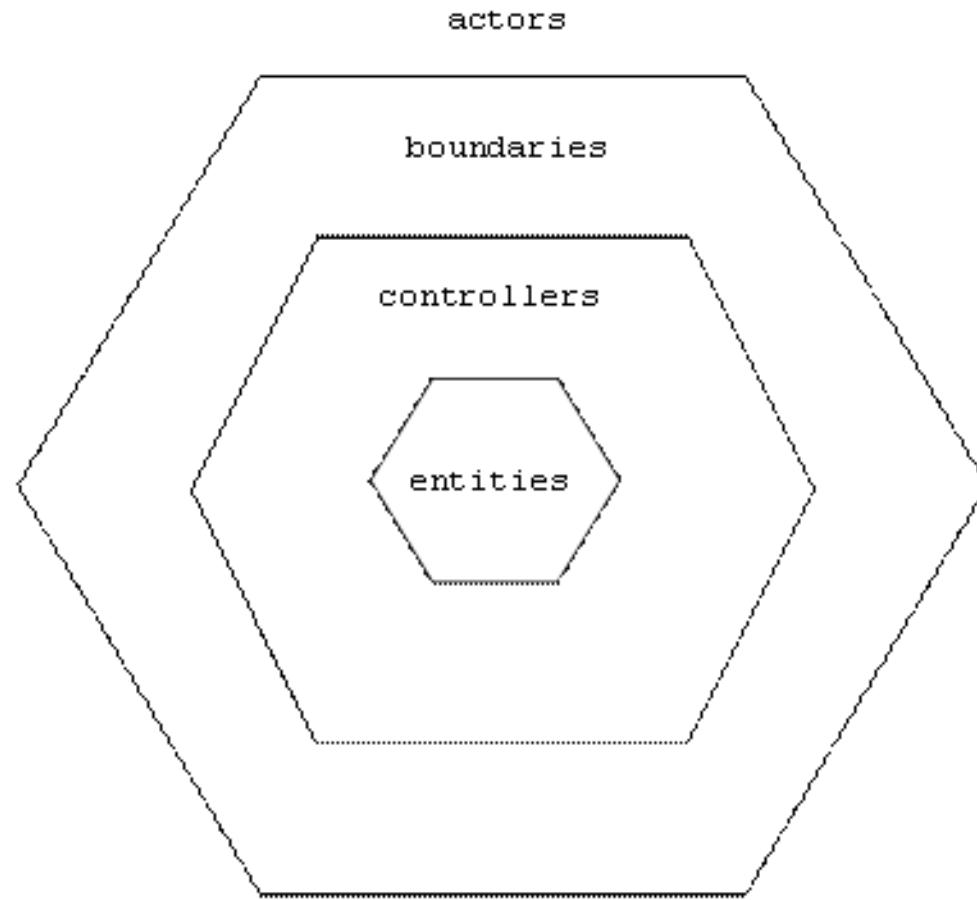


Entity classes show the logical data structure, which will help us understand what the system is supposed to offer to its users.



# Control Classes

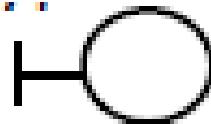
- Control objects are responsible for application specific business logic
- In addition, these object types also function as an intermediary between the system's various boundary and entity objects
- Within the context of use case realization, each boundary class will communicate with a single control class and control classes will be used to manage each use case's flow of execution



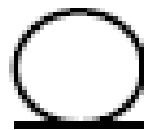
# Stereotypes and Classes

A Stereotype is a mechanism use to categorize classes.

- Primary class stereotypes in UML



*Boundary*



*Entity*



*Control*

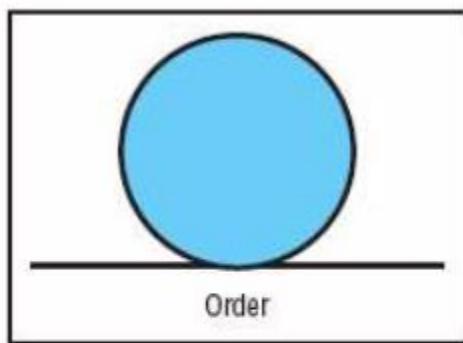


# Analysis Classes

- Entity class :
  - Persistent data (used multiple times and in many UCs)
  - Still exists after the UC terminates (e.g. DB storage)
- Boundary class:
  - (User) interface between actors and the system
  - E.g. a Form, a Window (Pane)
- Control class:
  - Encapsulates business functionality

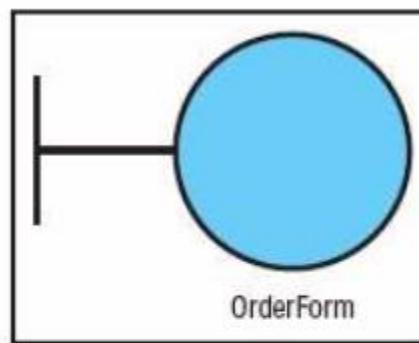
# Stereotypes of Analysis Classes

**Figure 9.3a**  
Entity Class Order



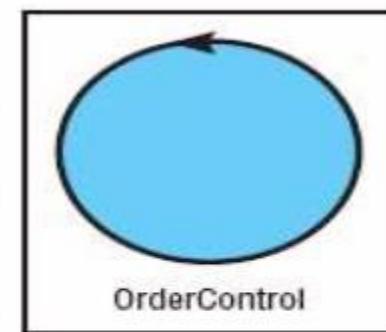
Mostly corresponds to conceptual data model classes

**Figure 9.3b**  
Boundary Class OrderForm



Encapsulates connections between actors and use cases

**Figure 9.3c**  
Control Class OrderControl



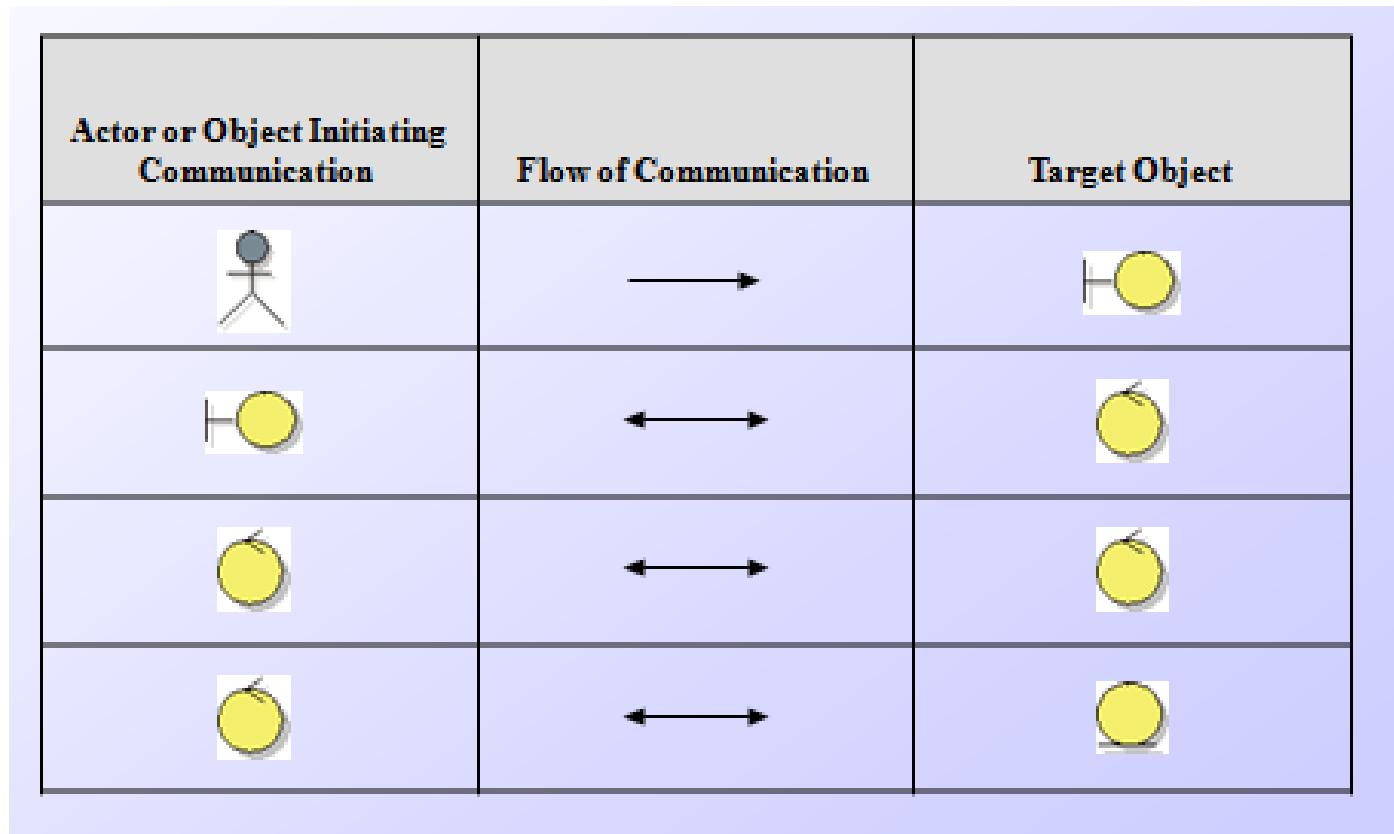
Mostly performs behaviors associated with inner workings of use cases



# Entity, Control, and Boundary Design Pattern

Stereotype	UML Element	Element in analysis class diagram	Icon in the Rational Unified Process
«boundary»	Class	Class with stereotype «boundary»	
«control»	Class	Class with stereotype «control»	
«entity»	Class	Class with stereotype «entity»	

# Allowed Communication within the ECB Design Pattern





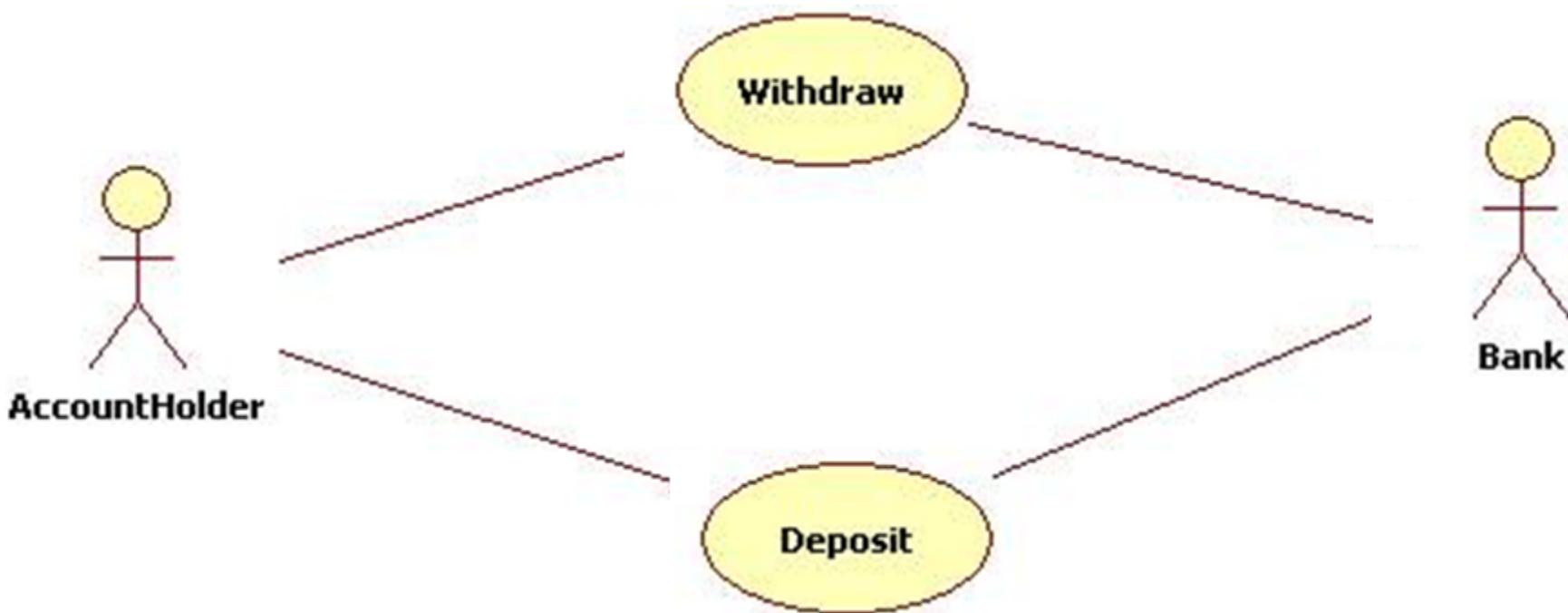
# What Is Robustness Analysis?

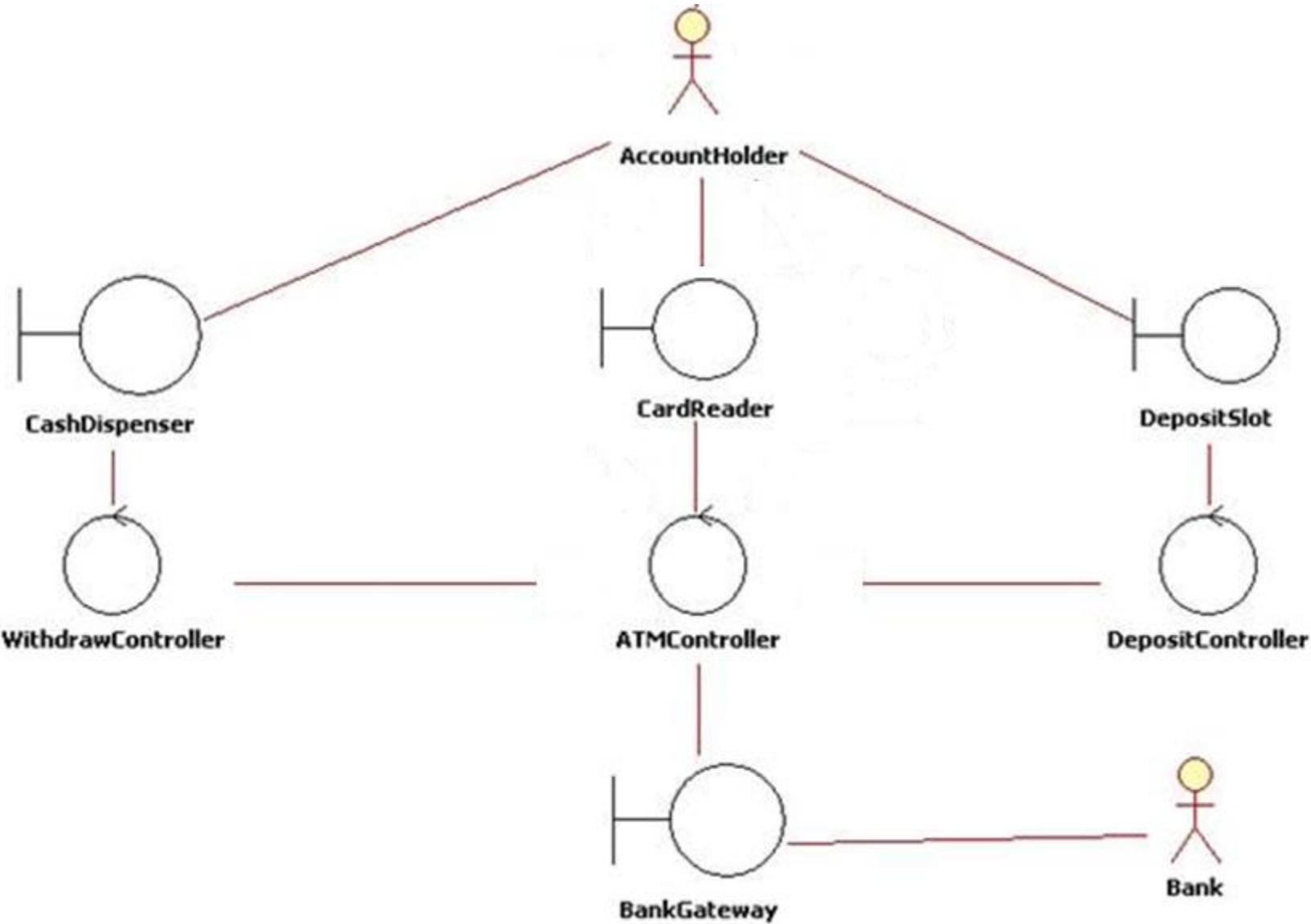
- Involves analyzing the narrative text of each of the use cases and identifying a first-guess set of the objects into entity, boundary, and control classes
- Requires completeness checks and adherence to diagramming rules



# ATM Scenario

- The Simple ATM allows account holders to make deposits to and withdraw funds from any accounts held at any branch of the Bank of Pakistan.





# Robustness Diagram- ECB

## Pattern(Example)

- The UNIVERSITY OF KARACHI registration system is briefly described thus:
- You have been asked to streamline, improve, and automate the process of **assigning professors to courses** and the **registration of students** such that it takes advantage of prevailing web technologies for on-line real time, location independent access.
- The process begins by professors deciding on which courses they will teach for the semester. The Registrar's office then enters the information into the computer system, allocating times, room, and student population restrictions to each course. A batch report is then printed for the professors to indicate which courses they will teach. A course catalogue is also printed for distribution to students.

# Robustness Diagram- ECB

## Pattern(Example)

- Students then select what courses they desire to take and indicate these by completing paper-based course advising forms. They then meet with an academic advisor who verifies their eligibility to take the selected courses, that the sections of the courses selected are still open, and that the schedule of selected courses does not clash. The typical student load is four courses.
- 
- The advisor then approves the courses and completed the course registration forms for the student. These are then sent to the registrar who keys them into the registration system – thereby formally registering a student. If courses selected are not approved, the student has to select other courses and complete the course advising forms afresh.



# Robustness Diagram- ECB Pattern(Example)

- Most times students get their first choice, however, in those cases where there is a conflict, the advising office talks with the students to get additional choices. Once all students have been successfully registered, a hard copy of the students' schedule is sent to the students for verification.
- Most student registrations are processed within a week, but some exceptional cases take up to two weeks to resolve.
- Once the initial registration period is over, professors receive a student roster for each class they are scheduled to teach.

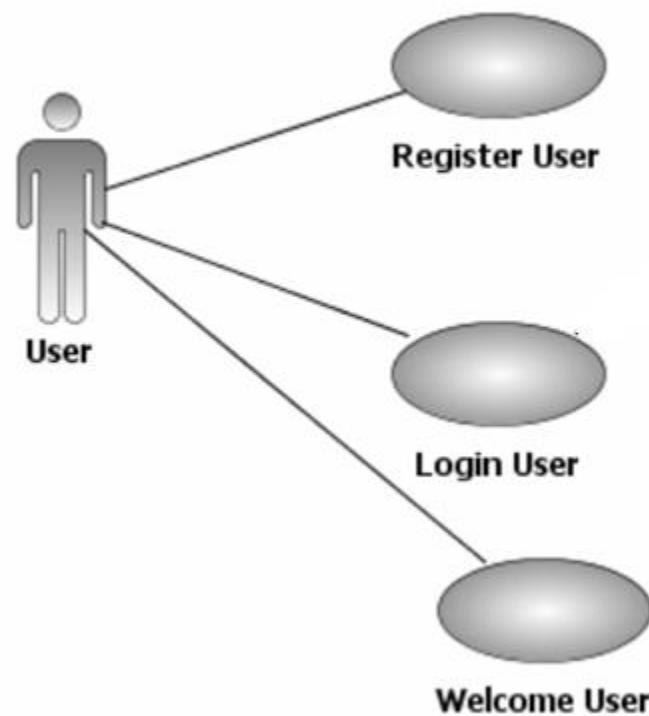


# Championship Management

- Design a system for organizing championships of table games (chess, go, backgammon, etc.)
  - Requirements:
    - A player should register and log in to the system before using it.
    - Each registered player may announce a championship.
    - Each player is allowed to organize a single championship at a time.
    - Players may join (enter) a championship on a web page – When the sufficient number of participants are present, the organizer starts the championship.
    - After starting a championship, the system must automatically create the pairings in a round-robin system.



# User Management (Design the ECB Diagram for the Use case given below)



# PROCESS MODEL

Lecture 8

# What is a Process Model ?

*It is a **description** of*

- i) what tasks need to be performed in*
- ii) what sequence under*
- iii) what conditions by*
- iv) whom to*

*achieve the “desired results.”*

## Why Have A Process Model?

- Provide “guidance” for a systematic coordination and controlling of
  - a) the tasks and of
  - b) the personnel who perform the tasks

**Note the key words:** coordination/control, tasks, people

# Extending the “Simple” Process

As projects got *larger* and more *complex*.

- Needed to **clarify and stabilize the requirements**
- Needed to **test more functionalities**
- Needed to **design more carefully**
- Needed to **use more existing software & tools**
  - Database
  - Network
  - Code control
- Needed **more people** to be involved

***Resulting in more tasks and more people***

# Effectiveness of using Correct Process Model

By changing the process model, we can improve :

- *Development speed (time to market)*
- *Product quality*
- *Project visibility*
- *Administrative overhead*
- *Risk exposure*
- *Customer relations, etc.*

Normally, a process model covers the entire  
**lifetime of a product.**

From **birth of a commercial idea**  
to **final installation of last release**

## Common Activities of Process Model

Many different software processes but all involve:

- **Specification** – defining what the system should do;
- **Design and implementation** – defining the organization of the system and implementing the system;
- **Validation** – checking that it does what the customer wants;
- **Evolution** – changing the system in response to changing customer needs.

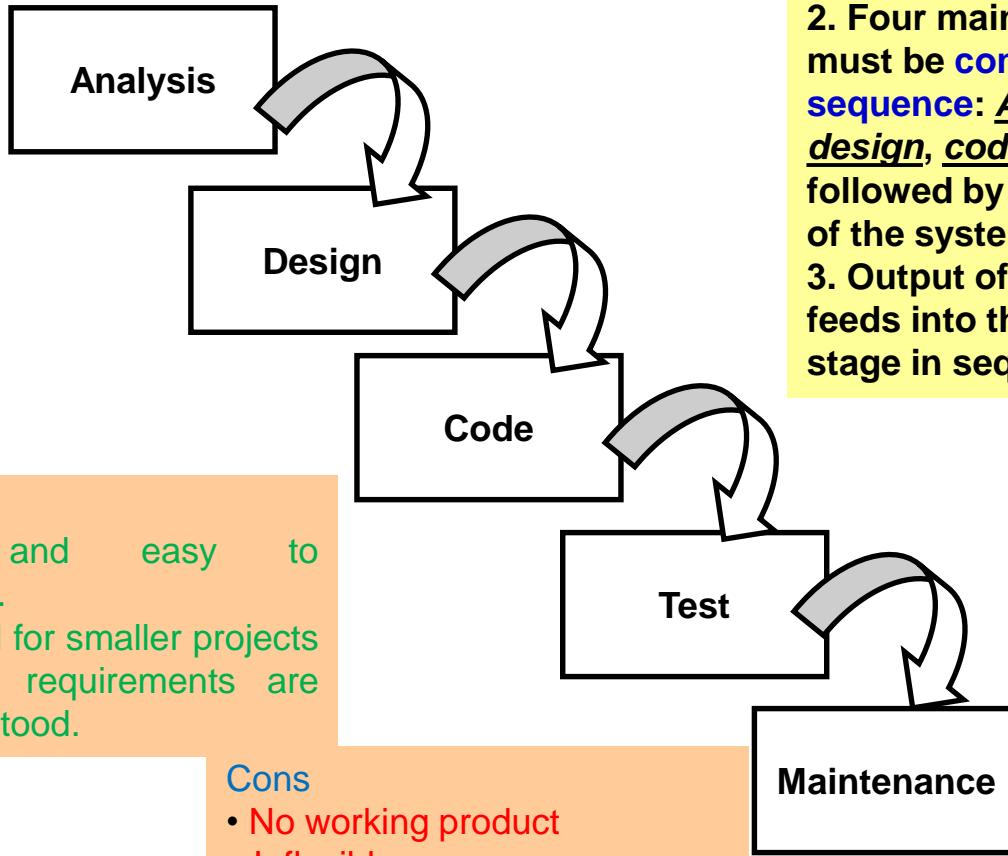
## Waterfall Model

### Pros

- Simple and easy to understand.
- Works well for smaller projects where the requirements are well-understood.

### Cons

- No working product
- Inflexible
- Poor model for large projects.



# Iteration of System Development Activities

## Problems

Lack of process visibility

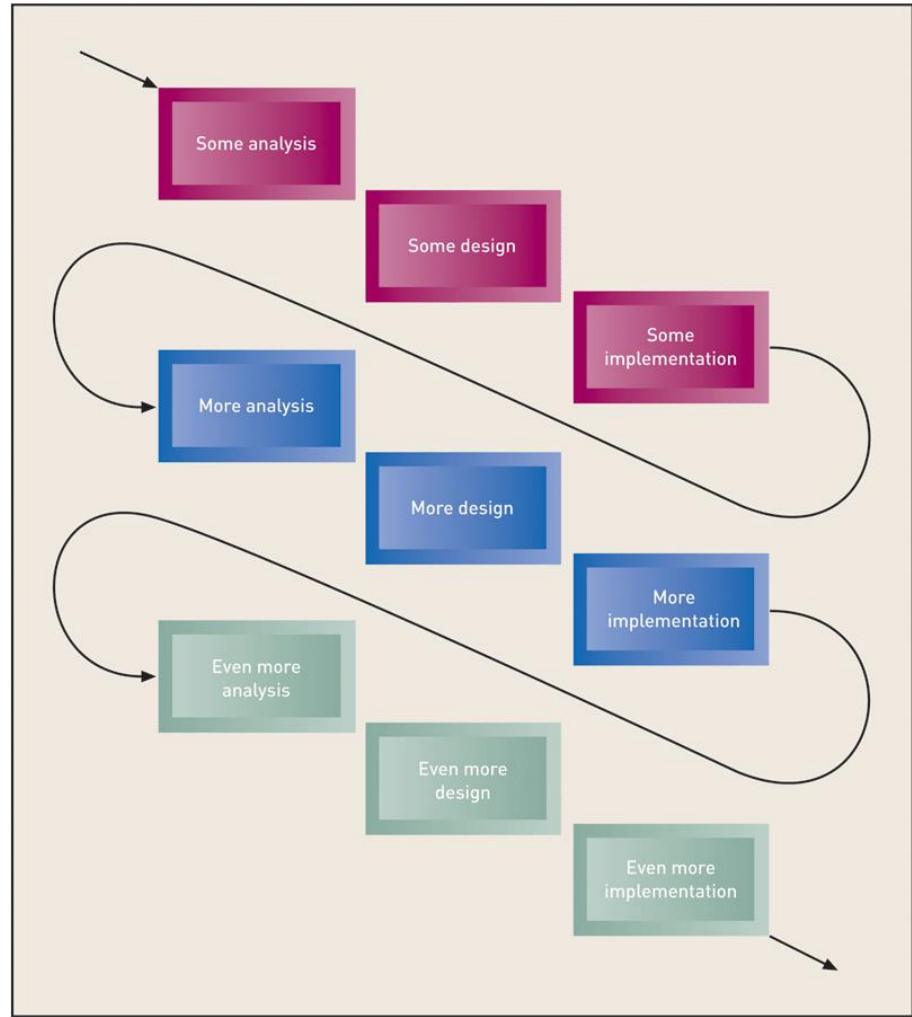
Systems are often poorly structured

## Applicability

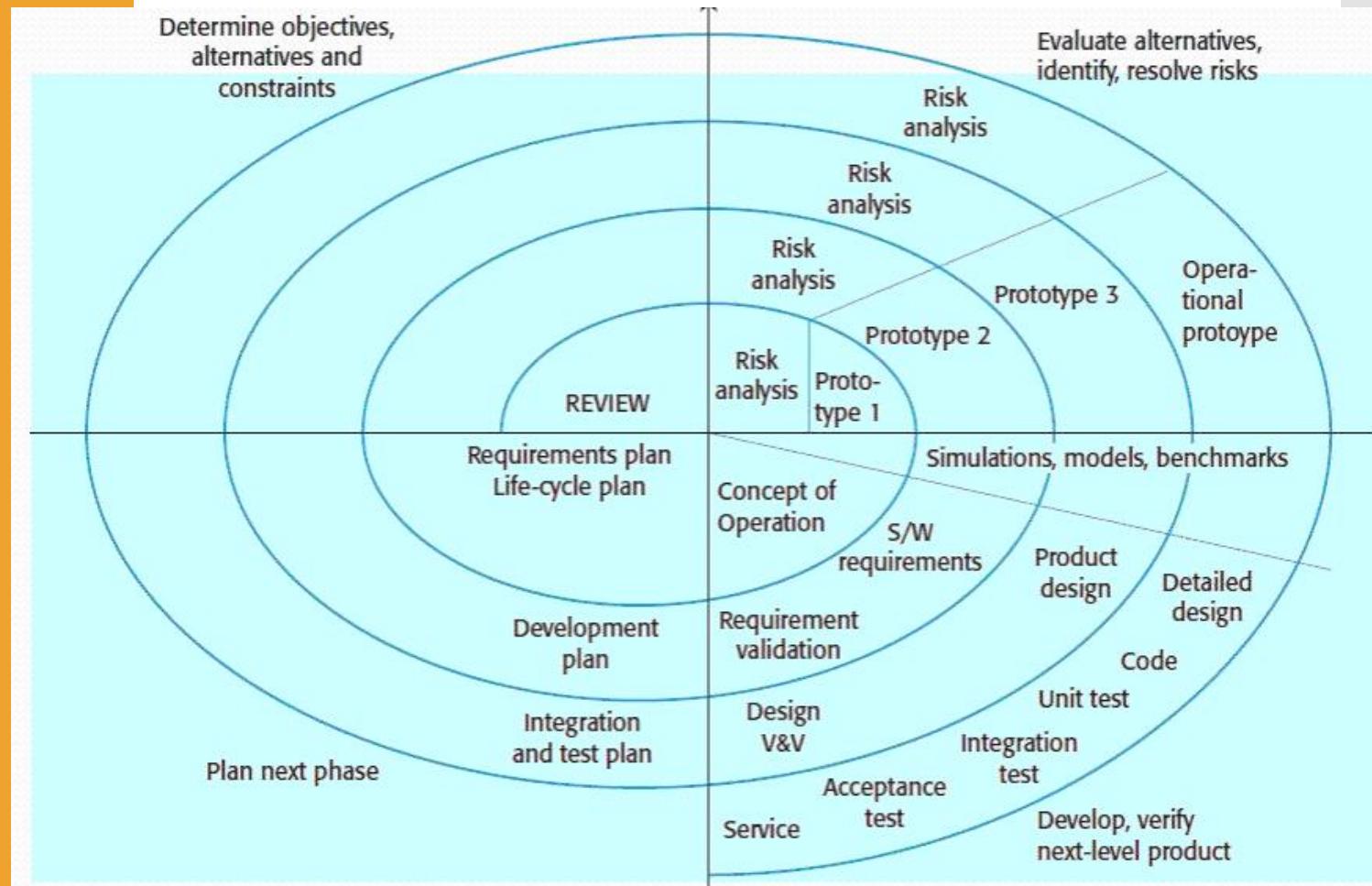
For small or medium-size interactive systems

For parts of large systems

For short-lifetime systems



# Iteration Model



# Agile Model

# Definition

**The requirements and solutions evolve through collaboration between self-organizing, cross-functional teams.**

**Break tasks into small increments with minimal planning and do not directly involve long-term planning.**

**Iterations are short time frames (time boxes) that typically last from one to four weeks.**

# Agile

- ⑩ Each iteration involves a *cross functional* team working in all functions:
  - Planning, and requirements analysis
  - Design, and coding
  - unit testing and acceptance testing
- ⑩ At the end of the iteration a **working product** is demonstrated to stakeholders.

# Characteristics of Agile Mode

**Iterative & Incremental**

**Test Driven Development**

**People Oriented**

**Ligtweight**

# UNIFIED PROCESSING

# Unified Processing

## What is an Unified Processing?

A process model that was created 1997 to give a framework for Object-oriented Software Engineering

Iterative, incremental model to adapt to specific project needs

The Unified Process is an adaptable methodology

The Unified Process is a modeling technique

# Unified Processing

## What are Characteristics of the Unified Process?

**Object Oriented**

**Use Case Driven**

**Architecture Centric**

**Iteration & Increment**

Focus core architecture in the early iterations

In earliest iterations, get high valued requirements

View of the whole design with the important characteristics made more visible

Expressed with classdiagram

- Iterations are time boxed (i.e. fixed in length)
- Best iteration length (2-6 weeks)

- Utilizes object oriented technologies.
- Classes are extracted during OOA and designed during OOD.

- Utilizes use case model to describe complete functionality of the system

# Unified Processing

What are the phases of UP?

Inception

Elaboration

Construction

Transition

# Inception

- A vision of the product is created. Questions discussed are:
- What is the product supposed to do?
- Why should my organization embark on a project to build this particular product?
- Does my organization have the resources to build this product?
- Is it feasible to do so?
- How much will this product cost and how much will it bring in?
- What will be the duration of the project?
- Risk analysis is performed.
- Decision whether to go ahead with the project or not is taken.

# Elaboration

- System to be built is analyzed in detail.
- Use cases used to document the requirements.  
Main aim: Get the core architecture and as many use cases as possible.
- The core architecture is coded, verified with user and baselined.
- Other high-risk requirements are identified and coded.
- A project plan is drawn in this phase, resources are allocated and a schedule is planned.
- UML diagrams are used to model the system under design.

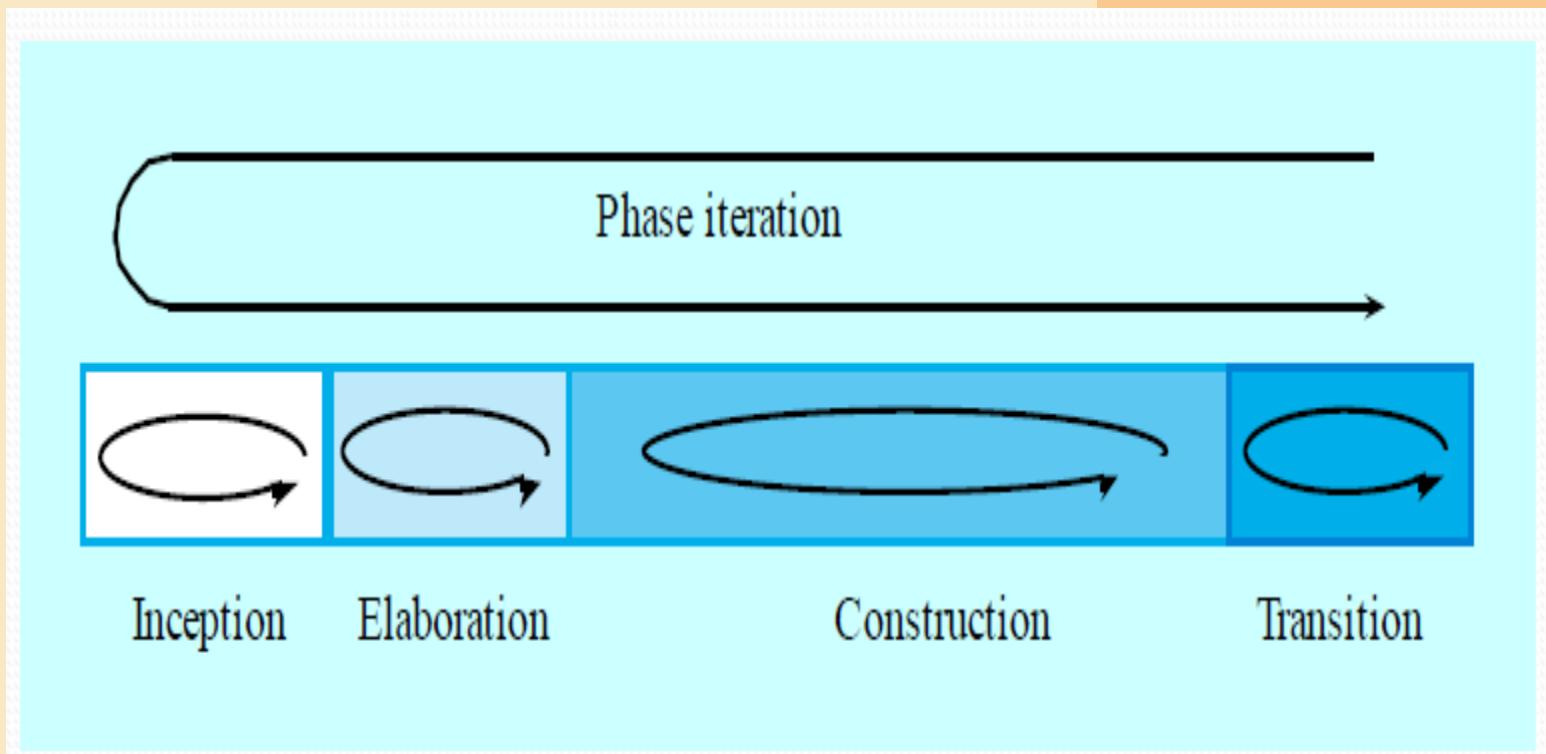
# Construction

- Remaining use cases are implemented.
- If any new use cases are discovered, they are implemented.
- Test cases are written, actual tests are carried out and a test report is prepared.
- Documentation for the system as well as guides for the users are written.

# Transition

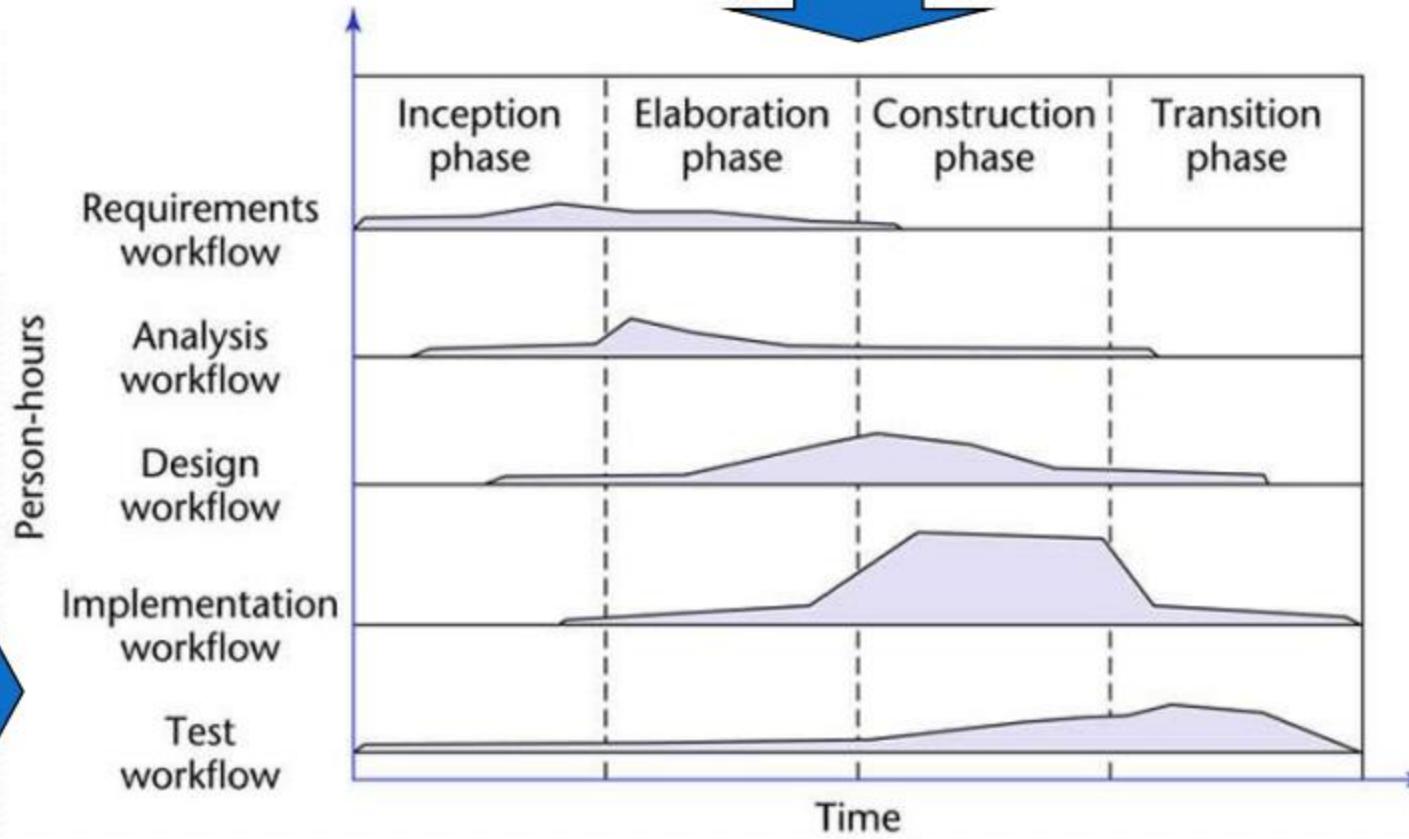
- System is installed in its environment and beta-tested.
- Feedback is received and System is refined and tuned to adapt in response to the feedback.
- It also includes activities like marketing of the product and training of users.

# Unified Process Model



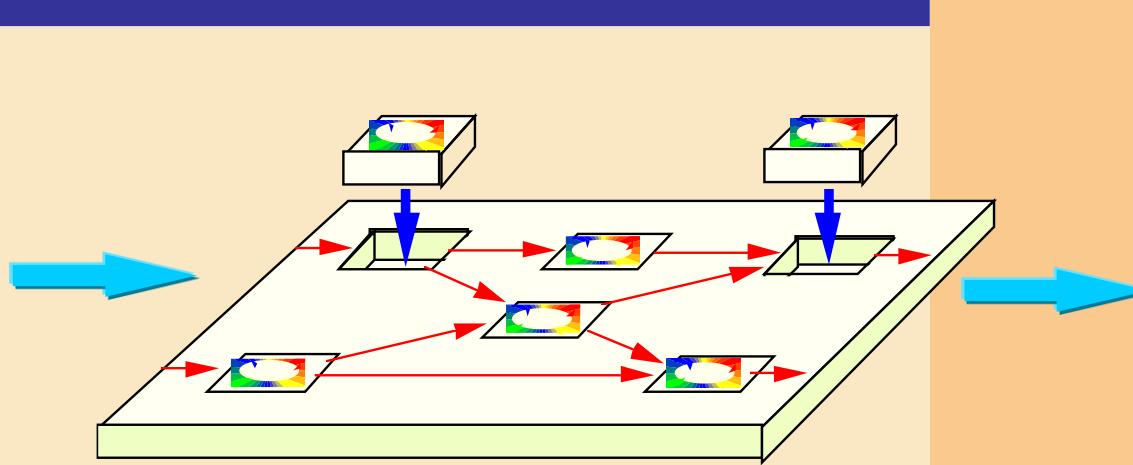
**Workflow** is  
Technical  
context of a step

•Phase is Business context of a step



Design Activities in the UP Life Cycle

# The Unified Process is a Process Framework



There is NO Universal Process!

- The Unified Process is designed for flexibility and extensibility
  - » allows a variety of lifecycle strategies
  - » selects what artifacts to produce
  - » defines activities and workers
  - » models concepts

# Repeated verification of quality

- The UP stresses on the repeated verification of quality.
- Here, quality refers to both, the quality of the software to be developed as well as the quality of the process itself.
- At the end of each iteration the team must produce executable software.
- This achievement indicates the success of the iteration.
- The software is tested and verified right from the start unlike the traditional SDLC where Quality Analysis is done basically at the end.

# Phase Deliverables

Inception Phase	Elaboration Phase	Construction Phase	Transition Phase
<ul style="list-style-type: none"><li>• The initial version of the domain model</li><li>• The initial version of the business model</li><li>• The initial version of the requirements artifacts</li><li>• A preliminary version of the analysis artifacts</li><li>• A preliminary version of the architecture</li><li>• The initial list of risks</li><li>• The initial ordering of the use cases</li><li>• The plan for the elaboration phase</li><li>• The initial version of the business case</li></ul>	<ul style="list-style-type: none"><li>• The completed domain model</li><li>• The completed business model</li><li>• The completed requirements artifacts</li><li>• The completed analysis artifacts</li><li>• An updated version of the architecture</li><li>• An updated list of risks</li><li>• The project management plan (for the rest of the project)</li><li>• The completed business case</li></ul>	<ul style="list-style-type: none"><li>• The initial user manual and other manuals, as appropriate</li><li>• All the artifacts (beta release versions)</li><li>• The completed architecture</li><li>• The updated risk list</li><li>• The project management plan (for the remainder of the project)</li><li>• If necessary, the updated business case</li></ul>	<ul style="list-style-type: none"><li>• All the artifacts (final versions)</li><li>• The completed manuals</li></ul>

# Examples Role in UP

- ⑩ **Stake Holder**
  - Customer, product manager, etc.
- ⑩ **Software Architect**
  - Establish and maintain architectural vision
- ⑩ **Process Engineer**
  - Leads definition and refinement of development use cases
- ⑩ **Graphics Artist**
  - Assists in user interface design

# Some UP guidelines

- Attack risks early on and continuously so, before they will attack you
- Stay focused on *developing executable software* in early iterations
- Prefer component-oriented architectures and reuse existing components
- Quality is a way of life, not an afterthought

# Six best “must” UP practices

1. Time-boxed iterations: *avoid speculative powerpoint architectures*
2. *Strive for cohesive architecture* and reuse existing components:
  - e.g. core architecture developed by small, co-located team
  - then early team members divide into sub-project leaders

# Six best “must” UP practices

3. Continuously verify quality: *test early & often*, and realistically by integrating all software at each iteration
4. Visual modeling: prior to programming, do at least some visual modeling to explore creative design ideas
5. Manage requirements: find, organize, and track requirements through skillful means
6. Manage change:
  - disciplined configuration management protocol, version control,
  - change request protocol
  - baselined releases at iteration ends

# WORKFLOWS

# The Unified Process

## Two Types

### Phases

Describe the business steps needed to develop, buy, and pay for software development.

The business increments are identified as phases

### WorkFlow

Describe the tasks or activities that a developer performs to evolve an information system over time

# WorkFlows

Workflow	Description
Business modelling	The business processes are modelled using business use cases.
Requirements	Actors who interact with the system are identified and use cases are developed to model the system requirements.
Analysis and design	A design model is created and documented using architectural models, component models, object models and sequence models.
Implementation	The components in the system are implemented and structured into implementation sub-systems. Automatic code generation from design models helps accelerate this process.
Test	Testing is an iterative process that is carried out in conjunction with implementation. System testing follows the completion of the implementation.
Deployment	A product release is created, distributed to users and installed in their workplace.
Configuration and change management	This supporting workflow managed changes to the system
Project management	This supporting workflow manages the system development
Environment	This workflow is concerned with making appropriate software tools available to the software development team.

# Primary Workflows

## ⑩ The Unified Process

### ⑩ PRIMARY WORKFLOWS

- Requirements workflow
- Analysis workflow
- Design workflow
- Implementation workflow
- Test workflow

### ⑩ Supplemental Workflows

- Planning Workflow

# Planning Workflow

- ⑩ Define scope of Project
- ⑩ Define scope of next iteration
- ⑩ Identify Stakeholders
- ⑩ Capture Stakeholders expectation
- ⑩ Build team
- ⑩ Assess Risks
- ⑩ Plan work for the iteration
- ⑩ Plan work for Project
- ⑩ Develop Criteria for iteration/project closure/success
- ⑩ UML concepts used: initial Business Model, using class diagram

# Requirements Workflow

- ⑩ The aim is to determine the client's needs
- ⑩ First, gain an understanding of the *domain* and build domain model describing important concepts of the context
- ⑩ Second, build a business model
  - Use UML to describe the client's business processes
  - If at any time the client does not feel that the cost is justified, development terminates immediately
- ⑩ It is vital to determine the client's constraints
  - Deadline -- Nowadays software products are often mission critical
  - Parallel running

# Analysis Workflow

- ⑩ The aim of the analysis workflow To analyze and refine the requirements
- ⑩ Two separate workflows are needed
  - The requirements artifacts must be expressed in the language of the client
  - The analysis artifacts must be precise, and complete enough for the designers
- ⑩ Specification document ("specifications") Constitutes a contract. It must not have imprecise phrases like "optimal," or "98 percent complete"
- ⑩ Having complete and correct specifications is essential for
  - Testing, and
  - Maintenance
- ⑩ The specification document must not have
  - Contradictions
  - Omissions
  - Incompleteness

# Design Workflow

- ⑩ **The goal is to refine the analysis workflow until the material is in a form that can be implemented by the programmers**
  - Many nonfunctional requirements need to be finalized at this time, including: Choice of programming language, Reuse issues, Portability issues.
- ⑩ **Classical Design**
- ⑩ **Architectural design**
  - Decompose the product into modules
- ⑩ **Detailed design**
  - Design each module using data structures and algorithms
- Object Oriented Design**
- ⑩ **Classes are extracted during the object-oriented analysis workflow, and**
  - Designed during the design workflow

# Implementation Workflow

- ⑩ The aim of the implementation workflow is to implement the target software product in the selected implementation language
- ⑩ Distribute the system by mapping executable components onto nodes in the deployment model
- ⑩ Implement Design Classes and subsystems through packaging mechanism:
- ⑩ Acquire external components realizing needed interfaces
- ⑩ Unit test the components
- ⑩ Integrate via builds

# Test Workflow

- ⑩ Carried out in parallel with other workflows
- ⑩ Primary purpose
  - To increase the quality of the evolving system
- ⑩ The test workflow is the responsibility of
  - Every developer and maintainer
  - Quality assurance group
- ⑩ Develop set of test cases that specify what to test in the system
  - many for each Use Case
  - each test case will verify one scenario of the use case
  - based on Sequence Diagram

# Deployment Workflow

- ⑩ Activities include
  - Software packaging
  - Distribution
  - Installation
  - Beta testing

The Customer clicks the Log In button on the Home Page. The system displays the Login Page. The Customer enters his or her user ID and password and then clicks the Log In button. The system validates the login information against the persistent Account data and then returns the Customer to the Home Page.

#### Alternate Courses

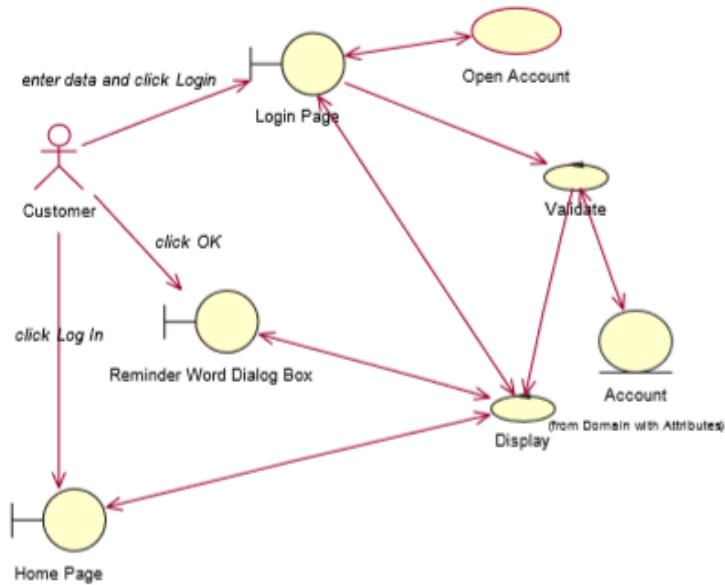
If the Customer clicks the New Account button on the Login Page, the system invokes the Open Account use case.

If the Customer clicks the Reminder Word button on the Login Page, the system displays the reminder word stored for that Customer, in a separate dialog box. When the Customer clicks the OK button, the system returns the Customer to the Login Page.

If the Customer enters a user ID that the system does not recognize, the system displays a message to that effect and prompts the Customer to either enter a different ID or click the New Account button.

If the Customer enters an incorrect password, the system displays a message to that effect and prompts the Customer to re-enter his or her password.

If the Customer enters an incorrect password three times, the system displays a page telling the Customer that he or she should contact customer service, and also freezes the Login Page.



# Use Case Diagrams

Tutorial

# What is a use case?

- A requirements analysis concept
- A case of a use of the system/product
- Describes the system's actions from the point of view of a user
- ***Tells a story***
  - A sequence of events involving
  - Interactions of a user with the system
- Specifies one aspect of the behavior of a system,  
***without specifying the structure of the system***
- Is oriented toward satisfying a user's goal

# How do we describe use cases?

- Textual or tabular descriptions
- User stories
- Diagrams

# Use Case Descriptions

- **actors** - something with a behavior or role, e.g., a person, another system, organization.
- **scenario** - a specific sequence of actions and interactions between actors and the system, a.k.a. a *use case instance*
- **use case** - a collection of related success and failure scenarios, describing actors using the system to support a goal.

# What is an Actor?

- Include all user roles that interact with the system
- Include system components only if they responsible for initiating/triggering a use case.
  - For example, a timer that triggers sending of an e-mail reminder
- **primary** - a user whose goals are fulfilled by the system
  - importance: define user goals
- **supporting** - provides a service (e.g., info) to the system
  - importance: clarify external interfaces and protocols
- **offstage** - has an interest in the behavior but is not primary or supporting, e.g., government
  - importance: ensure all interests (even subtle) are identified and satisfied

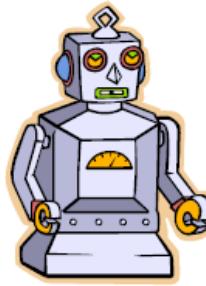
# Finding Actors [1]

External objects that produce/consume data:

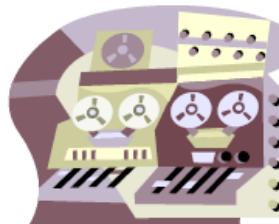
- Must serve as sources and destinations for data
- Must be external to the system



Humans



Machines



External systems



Organizational Units



Sensors

# Finding Actors [2]

**Ask the following questions:**

- Who are the system's primary users?
- Who requires system support for daily tasks?
- Who are the system's secondary users?
- What hardware does the system handle?
- Which other (if any) systems interact with the system in question?
- Do any entities interacting with the system perform multiple roles as actors?
- Which other entities (human or otherwise) might have an interest in the system's output?

# What is a user story?

- An abbreviated description of a use case
- Used in agile development

Answers 3 questions:

1. Who?
2. Does what?
3. And why?

*As a <type of user>,  
I want <some behavior from the system>  
so that <some value is achieved>*



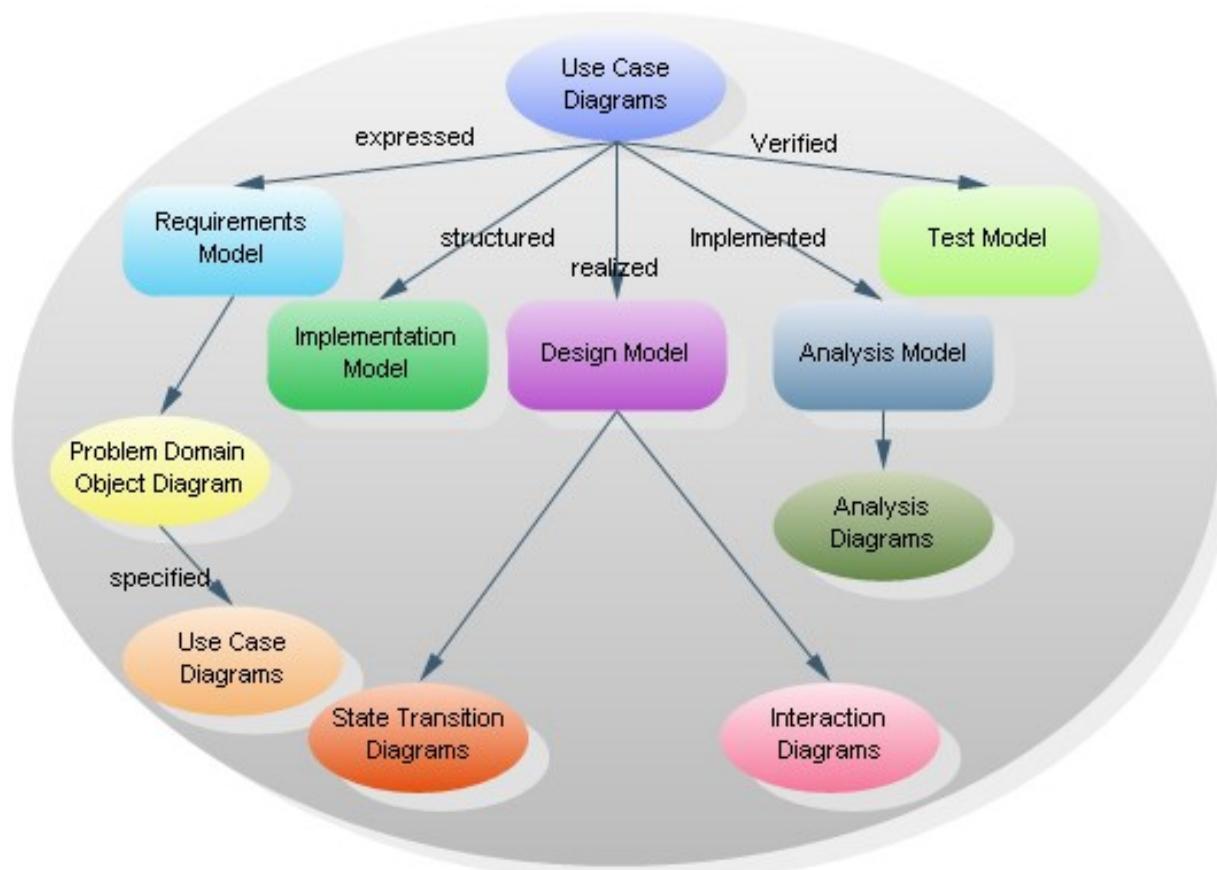
# Use Case Diagrams

- A picture
  - describes how actors relate to use cases
  - and use cases relate to one another
- Diagrams are not essential
- They are helpful in giving an overview, but only secondary in importance to the textual description
- They do not capture the full information of the actual use cases
- In contrast, text *is* essential

# Use Case Diagram **Objective**

- Built in early stages of development
- Purpose
  - Specify the context of a system
  - Capture the requirements of a system
  - Validate a systems architecture
  - Drive implementation and generate test cases
  - Developed by analysts and domain experts

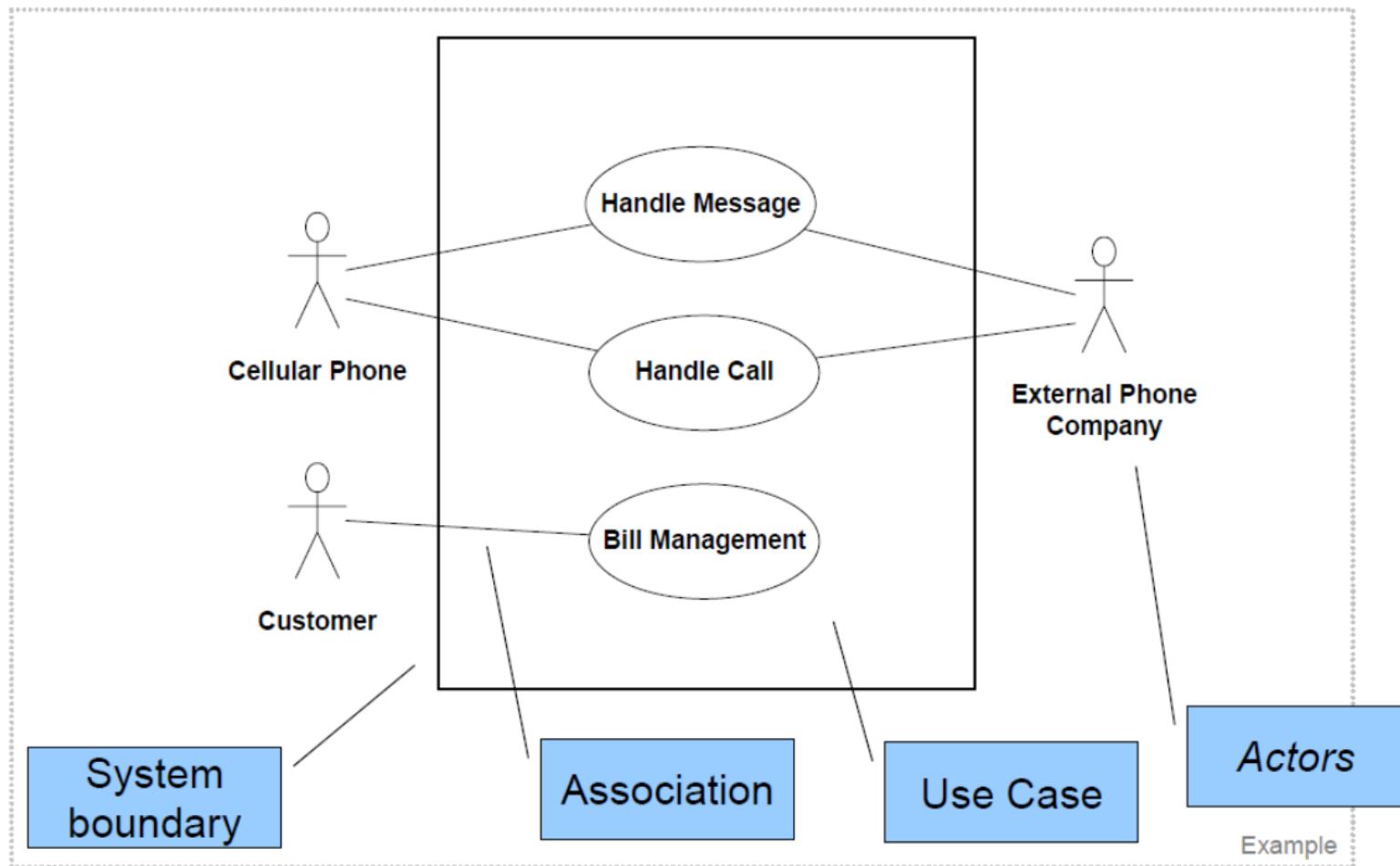
# How do use case diagrams fit in?



This applies also to use case descriptions.

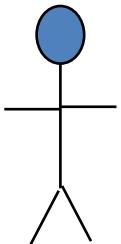
*Diagram reproduced from [www.edrawsoft.com](http://www.edrawsoft.com).*

# Example Use-Case Diagram



A standard form of use case diagram is defined in the Unified Modeling Language.

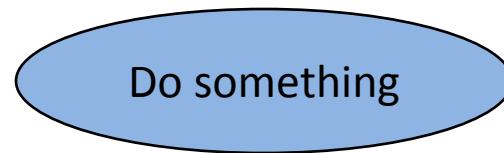
# Elements of use case diagram: Actor



Name

- Actor is someone interacting with use case (system function). Named by noun.
- Similar to the concept of user, but a user can play different *roles*; (example: a prof. can be instructor and researcher – plays 2 roles with two systems).
- Actor *triggers* use case.
- Actor has responsibility toward the system (inputs), and Actor have expectations from the system (outputs).

# Elements of use case diagram: Use Case



- System function (process – automated or manual).
- Named by verb.
- Each Actor must be linked to a use case, while some use cases may not be linked to actors.

USER/ACTOR	USER GOAL = Use Case
Order clerk	Look up item availability Create new order Update order
Shipping clerk	Record order fulfillment Record back order
Merchandising manager	Create special promotion Produce catalog activity report

# Elements of use case diagram: Other details

— Connection between Actor and Use Case



Boundary of system

<<include>>

**Include** relationship between Use Cases (one UC must call another; e.g., Login UC includes User Authentication UC)

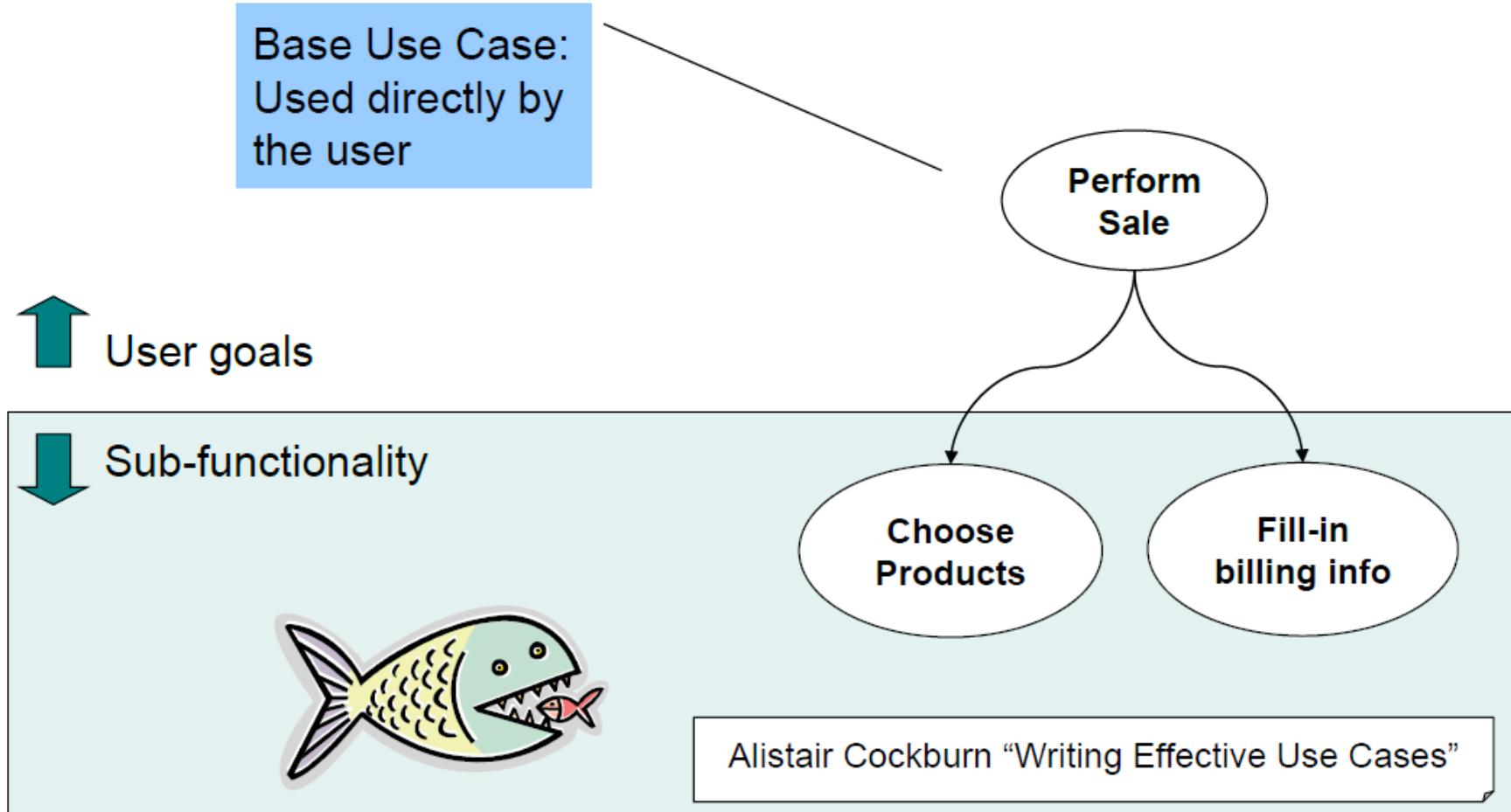
<<extend>>

**Extend** relationship between Use Cases (one UC calls Another under certain condition; think of if-then decision points)

# Linking Use Cases

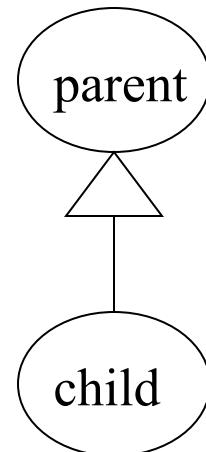
- *Association* relationships
- *Generalization* relationships
  - One element (child) "is based on" another element (parent)
- *Include* relationships
  - One use case (base) includes the functionality of another (inclusion case)
  - Supports re-use of functionality
- *Extend* relationships
  - One use case (extension) extends the behavior of another (base)

# Use Case Levels

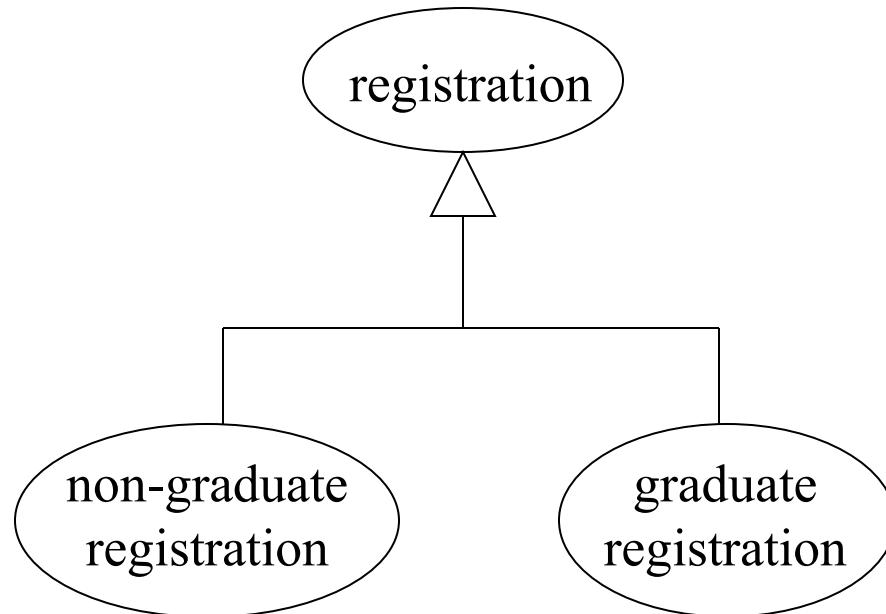


# 1. Generalization

- The child use case inherits the behavior and meaning of the parent use case.
- The child may add to or override the behavior of its parent.



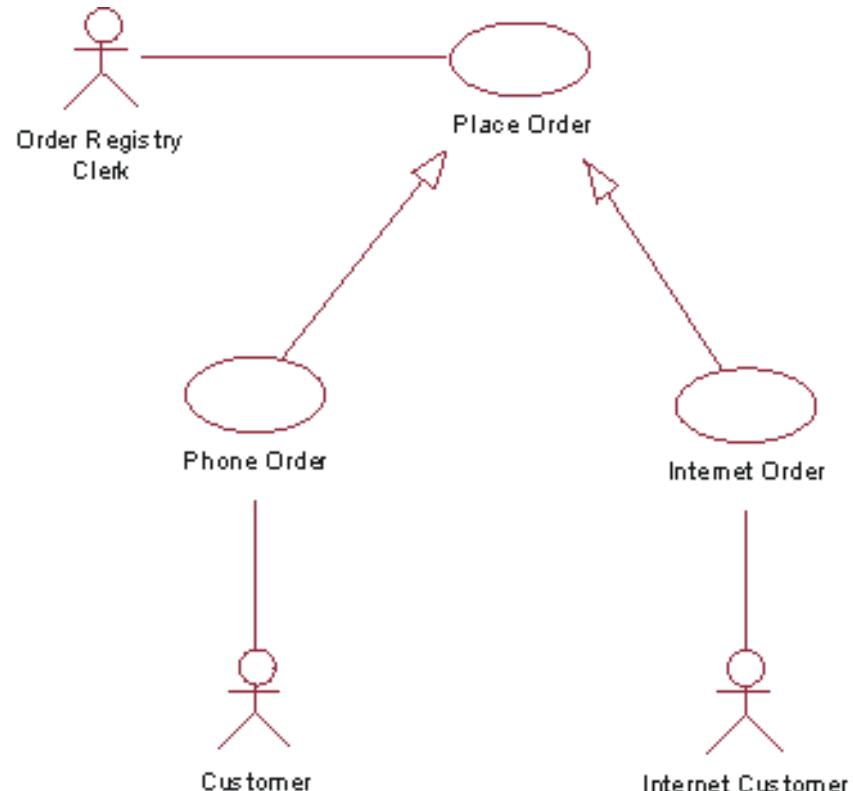
# More about Generalization



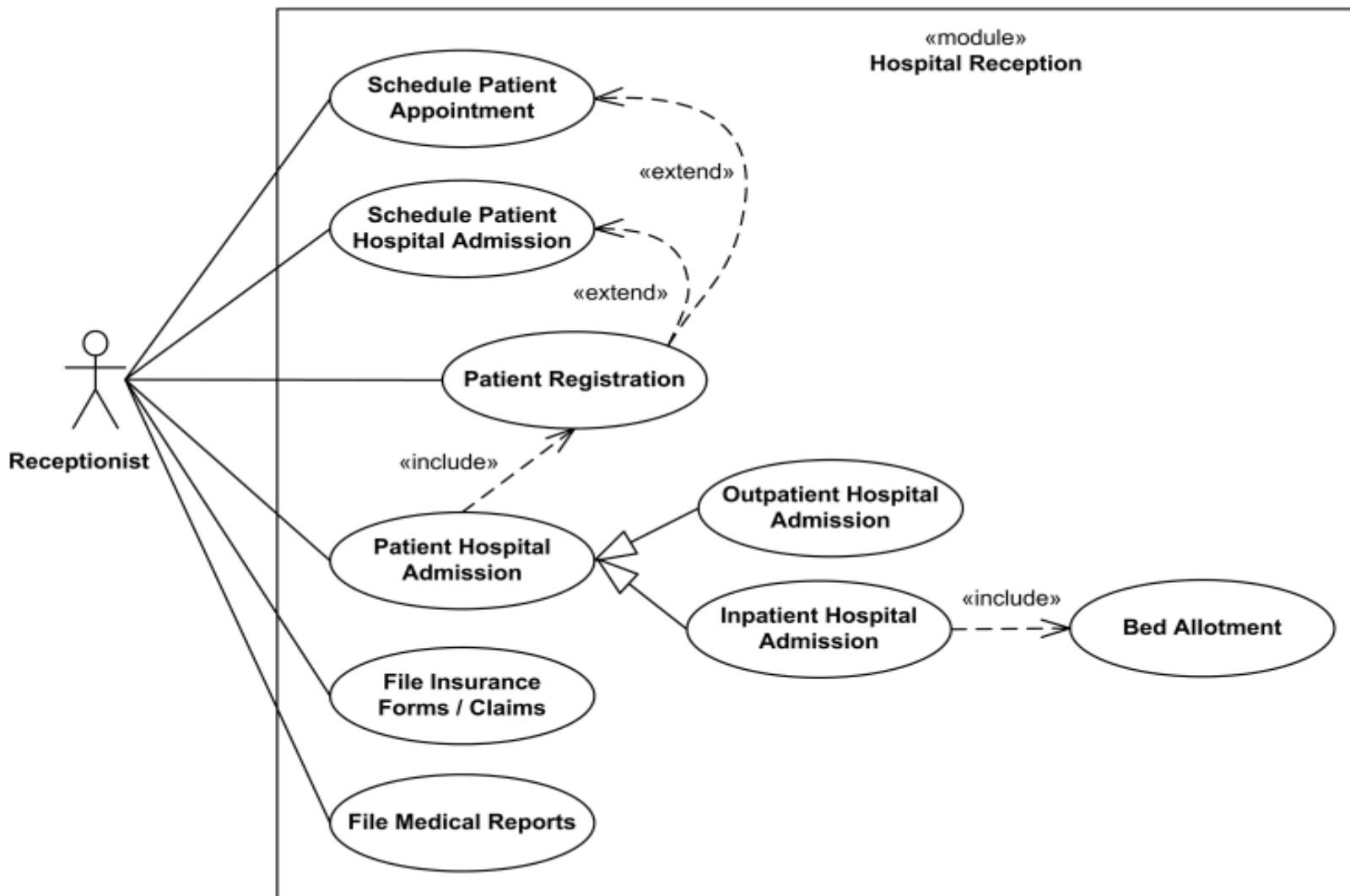
# Generalization Example

The actor Order Registry Clerk can instantiate the general use case Place Order.

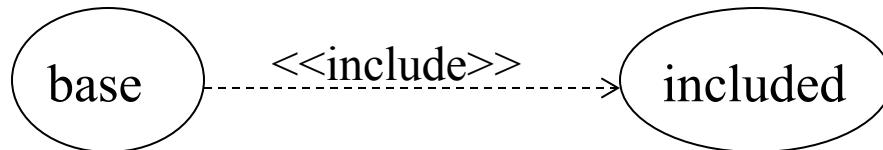
Place Order can also be specialized by the use cases Phone Order or Internet Order.



# Generalization Example



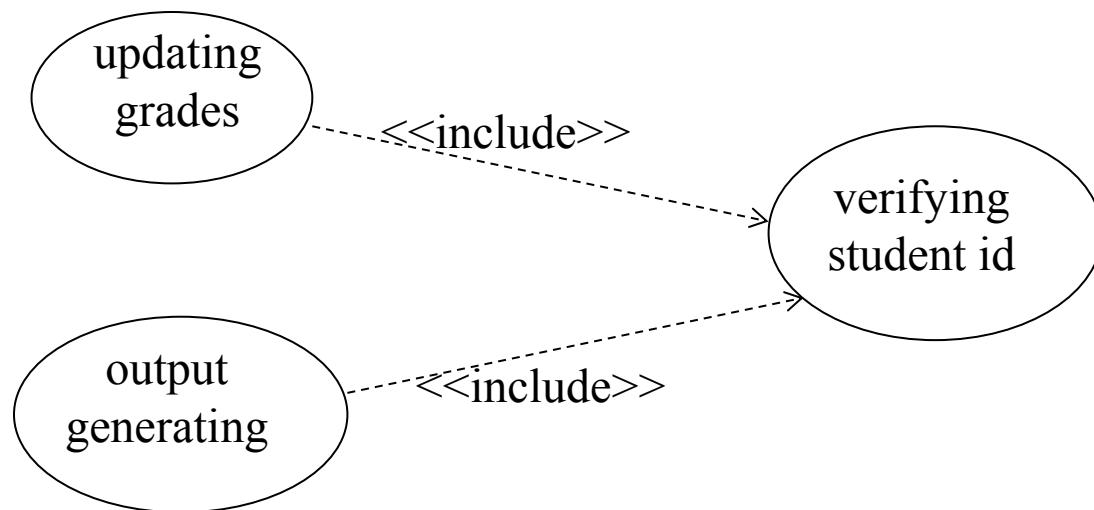
## 2. Include



- The base use case explicitly incorporates the behavior of another use case at a location specified in the base.
- The included use case never stands alone. It only occurs as a part of some larger base that includes it.

# More about **Include**

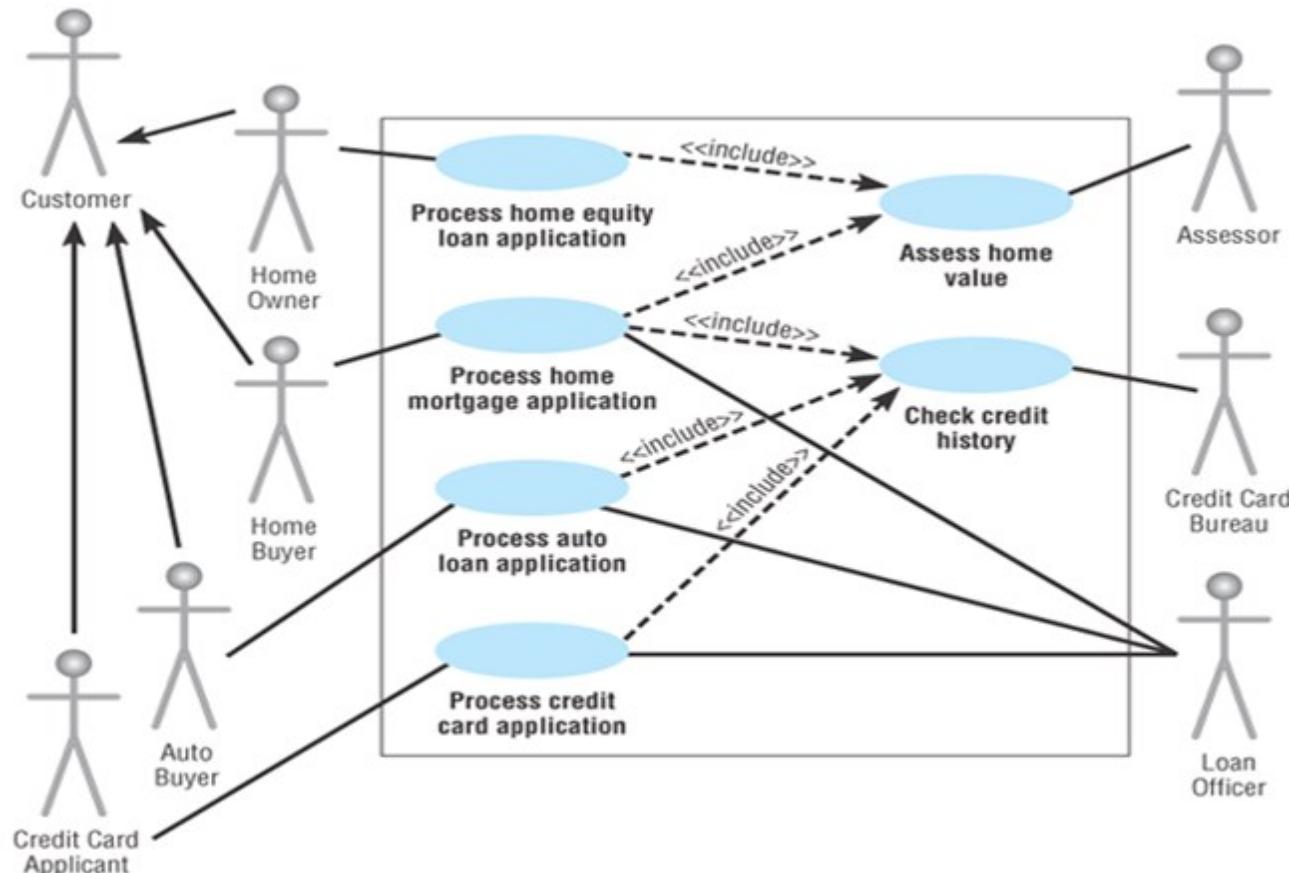
Enables us to avoid describing the same flow of events several times by putting the common behavior in a use case of its own.



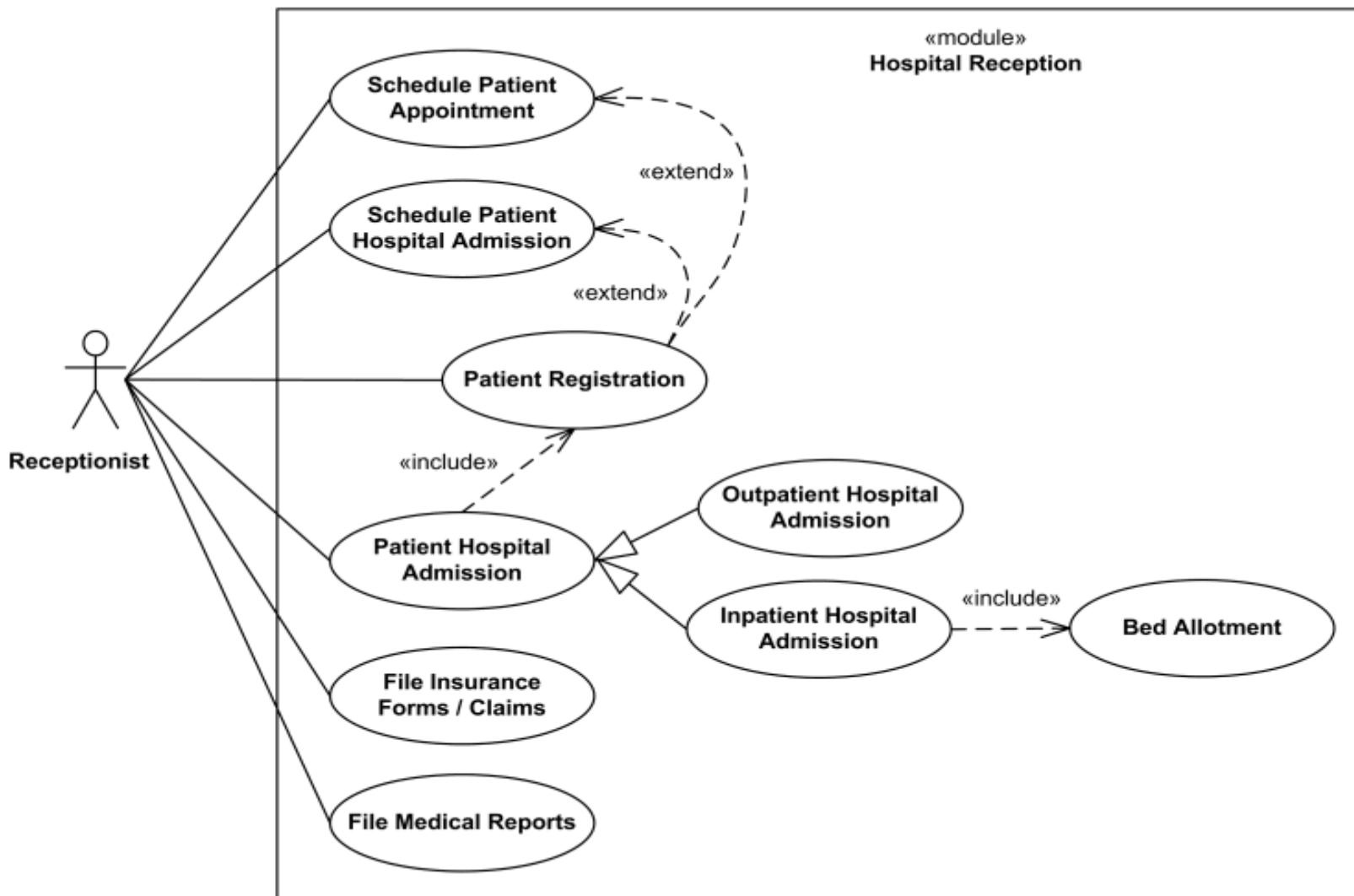
# Include relationship

- Include relationship – a standard case linked to a **mandatory** use case.
- Example: to *Authorize Car Loan* (standard use case), a clerk must run *Check Client's Credit History* (include use case).
- The standard UC includes the mandatory UC (use the verb to figure direction arrow).
- Standard use case can NOT execute without the include case → **tight coupling** .

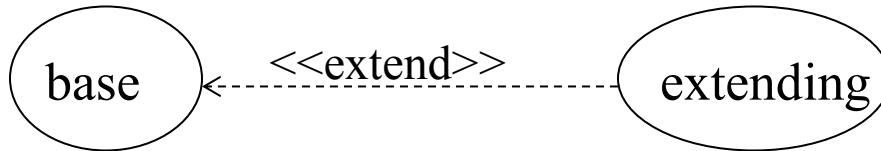
# Reading use case diagram with **Include** relationship



# Include Example



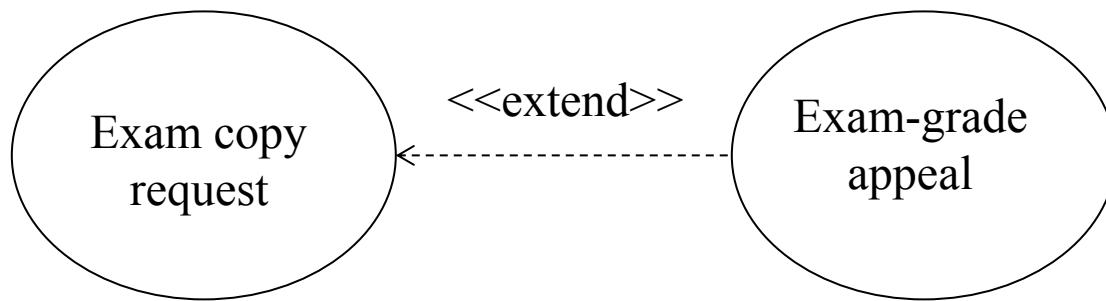
### 3. Extend



- The base use case implicitly incorporates the behavior of another use case at certain points called extension points.
- The base use case may stand alone, but under certain conditions its behavior may be extended by the behavior of another use case.

# More about Extend

- Enables to model optional behavior or branching under conditions.

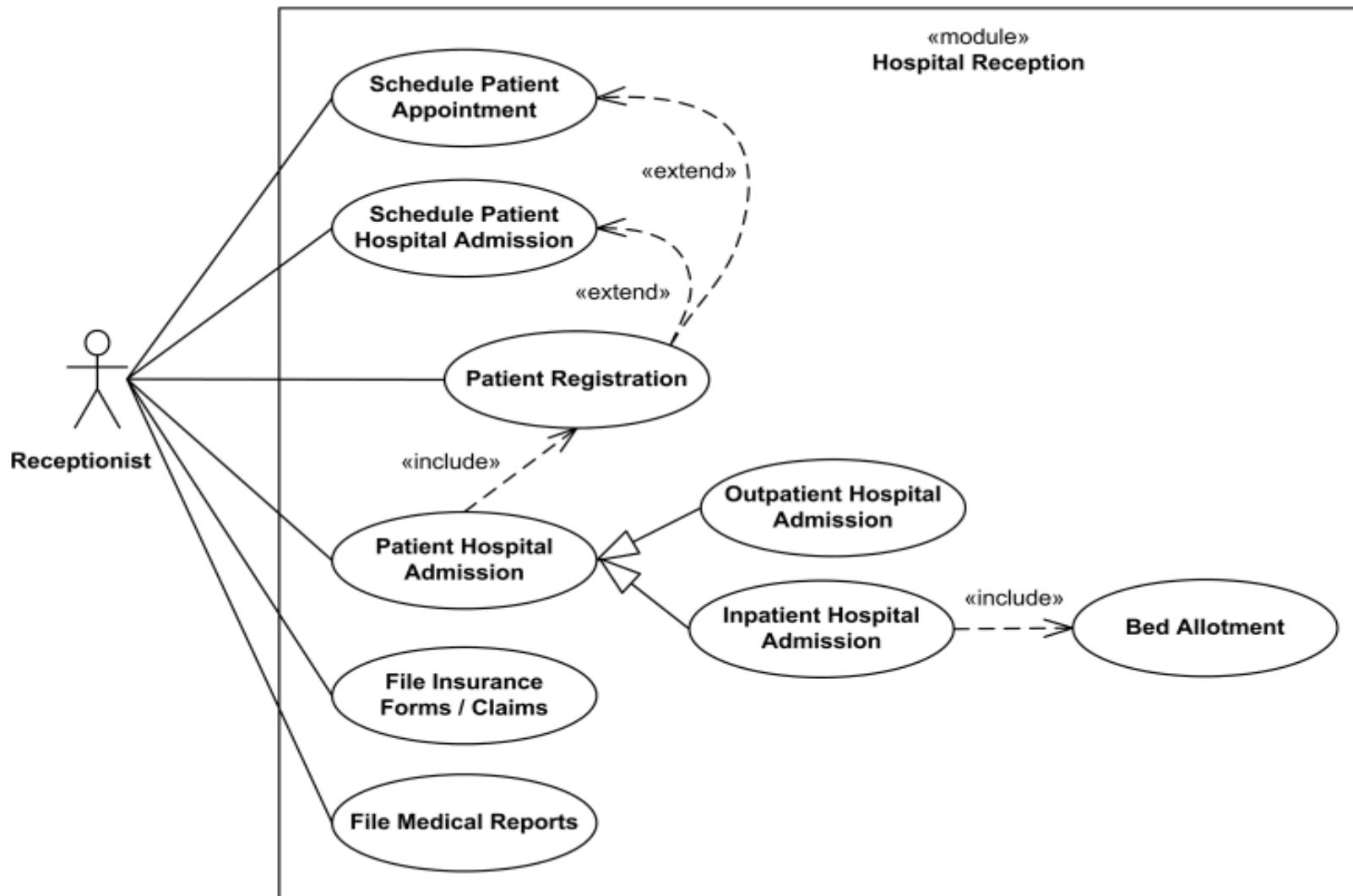


# **Extend** relationship

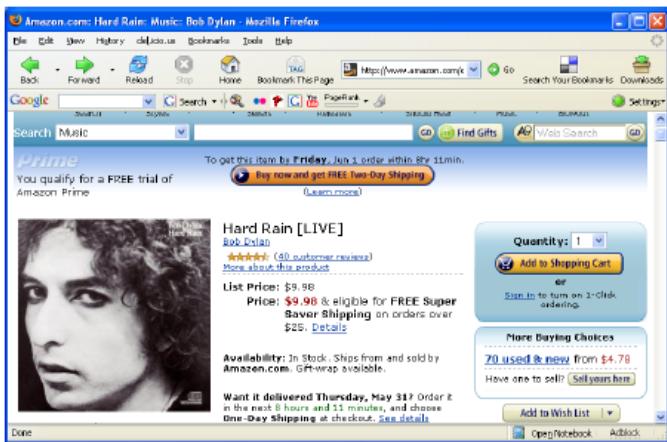
- Extend relationship – linking an ***optional*** use case to a standard use case.
- Example: *Register Course* (standard use case) may have *Register for Special Class* (extend use case) – class for non-standard students, in unusual time, with special topics, requiring extra fees...).
- The optional UC extends the standard UC
- Standard use case can execute without the extend case  
→ loose coupling.

[Reading extend relationship](#)

# Extend Example #1



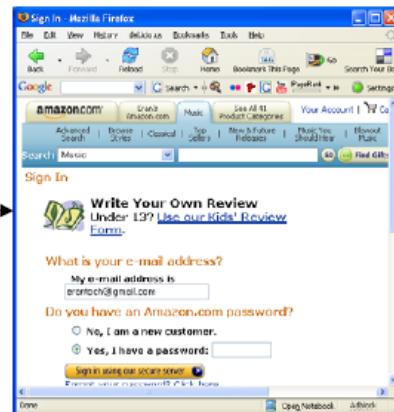
# Extend Example #2



Product Page

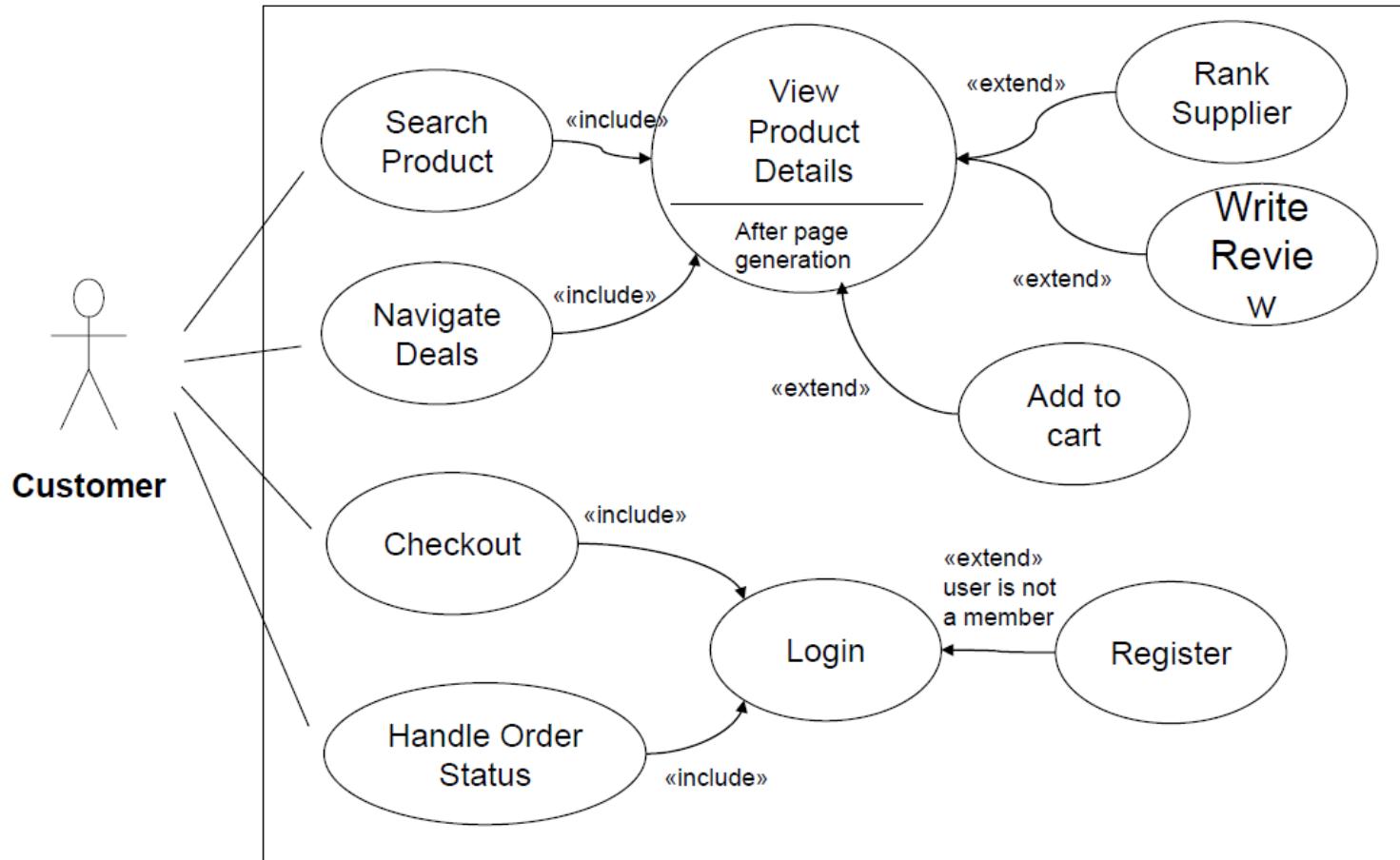


Shopping Cart



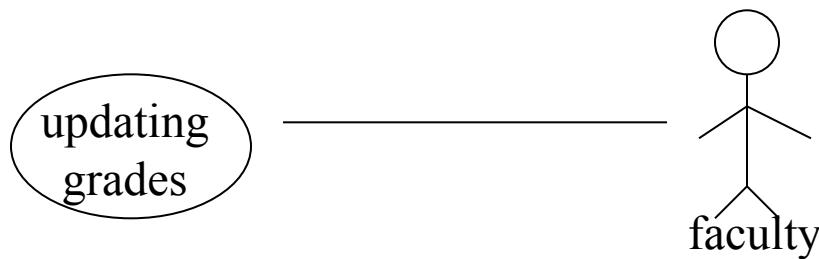
Review Writing

# Extend Example #2 cont.

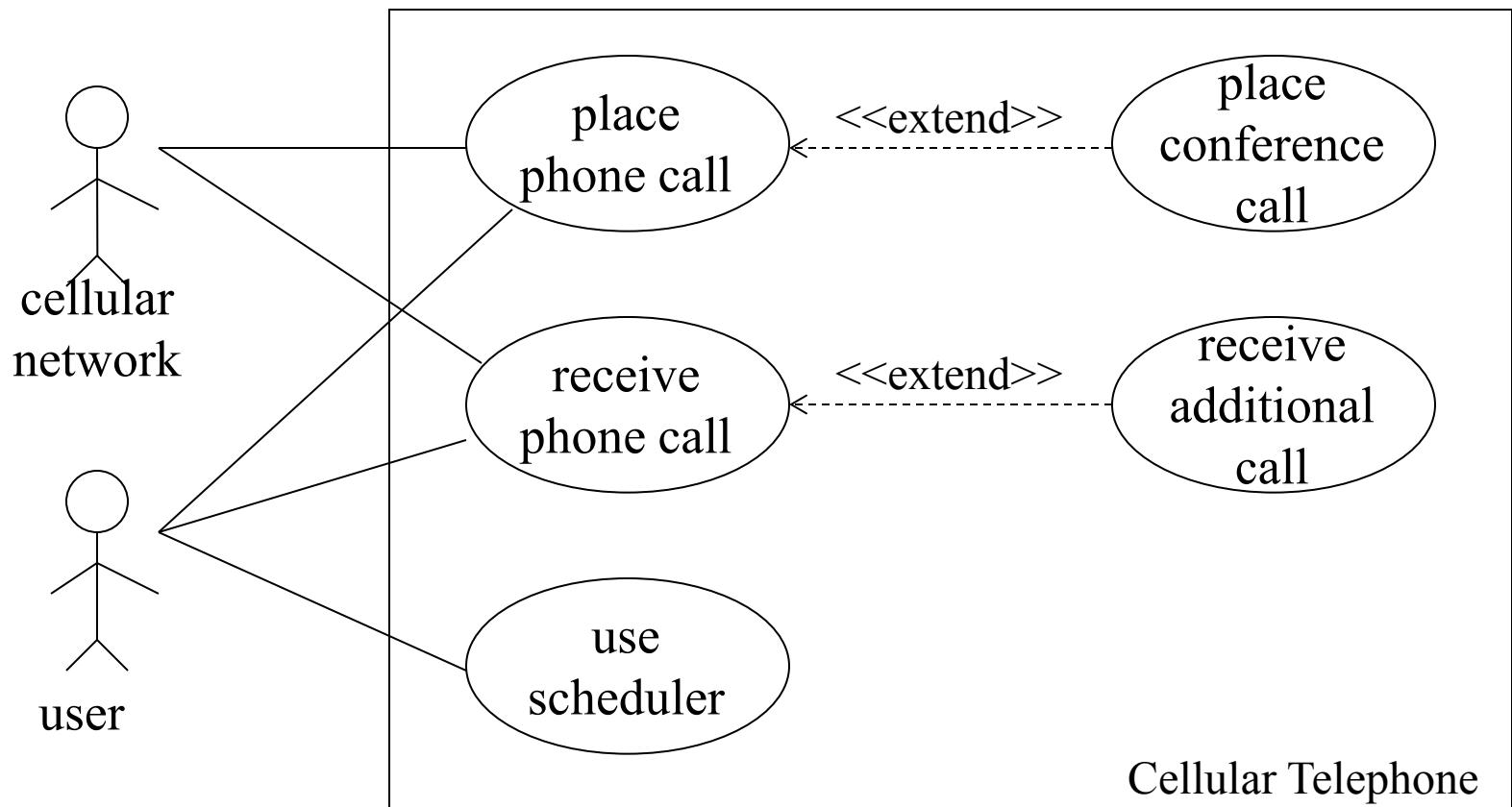


# Relationships between Use Cases and Actors

Actors may be connected to use cases by associations, indicating that the actor and the use case communicate with one another using messages.



# Example #1



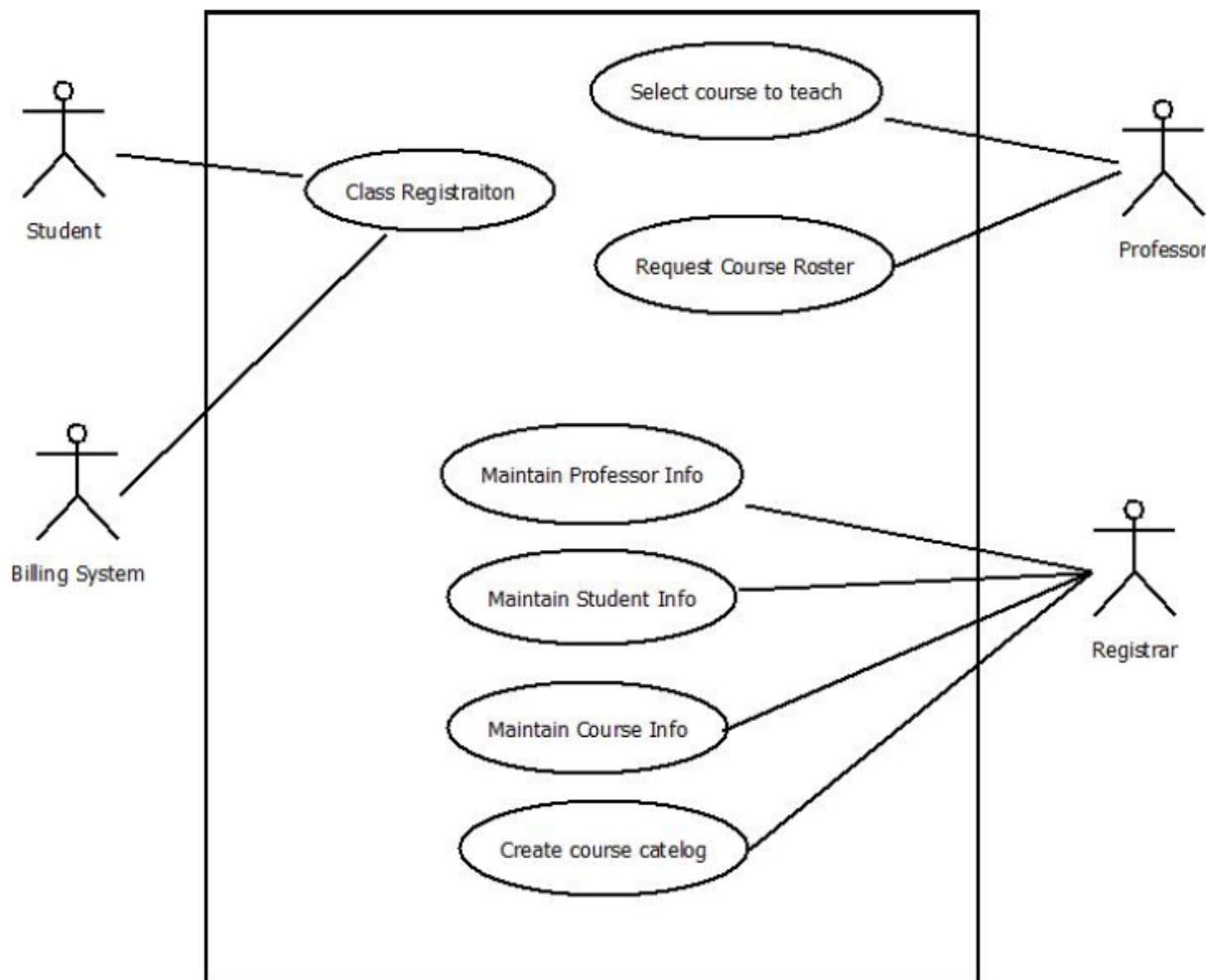
# Example #2

## Altered State University (ASU) Registration System

1. Professors indicate which courses they will teach on-line.
2. A course catalog can be printed
3. Allow students to select on-line four courses for upcoming semester.
4. No course may have more than 10 students or less than 3 students.
5. When the registration is completed, the system sends information to the billing system.
6. Professors can obtain course rosters on-line.
7. Students can add or drop classes on-line.

# Example #2 cont.

## Altered State University (ASU) Registration System



# How to create use case diagram

1. List main system functions (use cases) in a column:
  - think of business events demanding system's response
  - users' goals/needs to be accomplished via the system
  - Create, Read, Update, Delete (CRUD) data tasks
  - Naming use cases – user's needs usually can be translated in data tasks
2. Draw ovals around the function labels
3. Draw system boundary
4. Draw actors and connect them with use cases (if more intuitive, this can be done as step 2)
5. Specify include and extend relationships between use cases (yes, at the end - not before, as this may pull you into process thinking, which does not apply in UC diagramming).

# Use-Case Diagrams: Example [1]

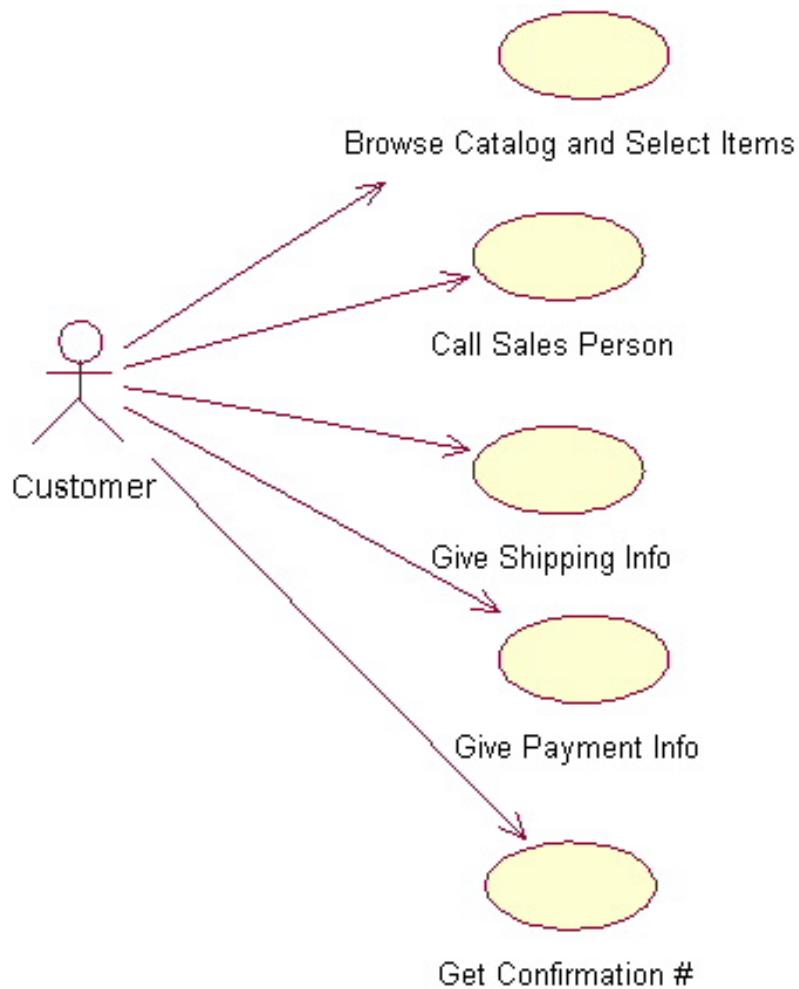
## I. Begin with a Use Case!

A user placing an order with a sales company might follow these steps :

1. Browse catalog and select items.
2. Call sales representative.
3. Supply shipping information.
4. Supply payment information.
5. Receive conformation number from salesperson.

## II. Then translate Use Case sequence into Diagram

# Use-Case Diagrams: Example [2]



The salesperson could also be included in this use case diagram because the salesperson is also interacting with the ordering system.

# Use-Case Diagram Case Study [1]

## Vending Machine

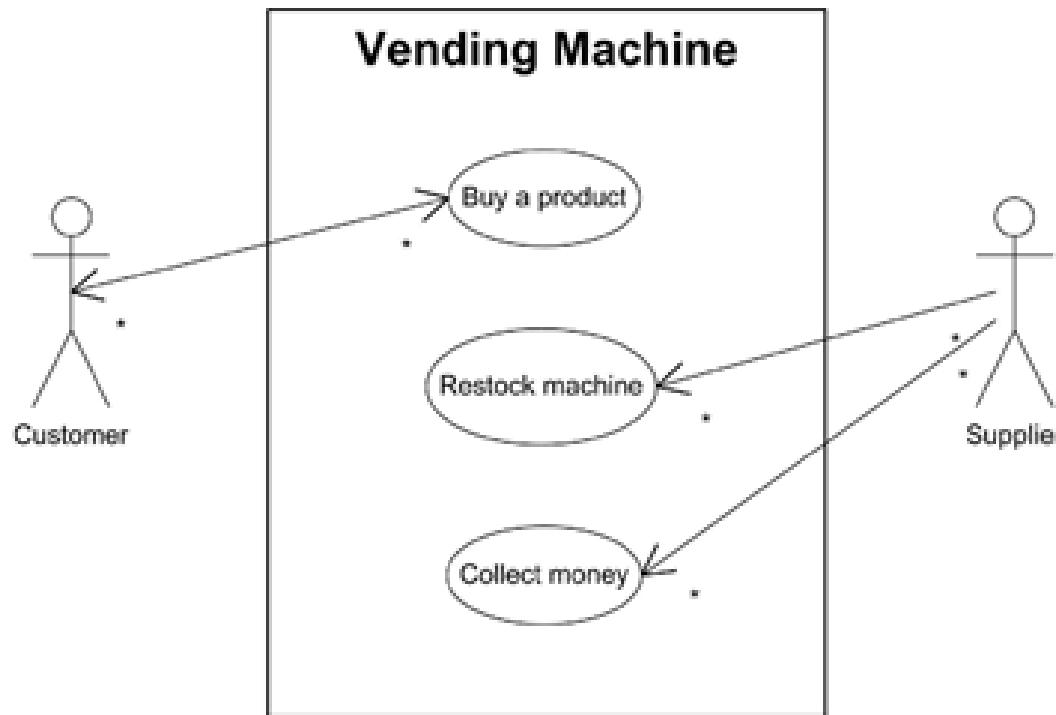
After client interview the following system scenarios were identified:

- A customer buys a product
- The supplier restocks the machine
- The supplier collects money from the machine

On the basis of these scenarios, the following three actors can be identified:

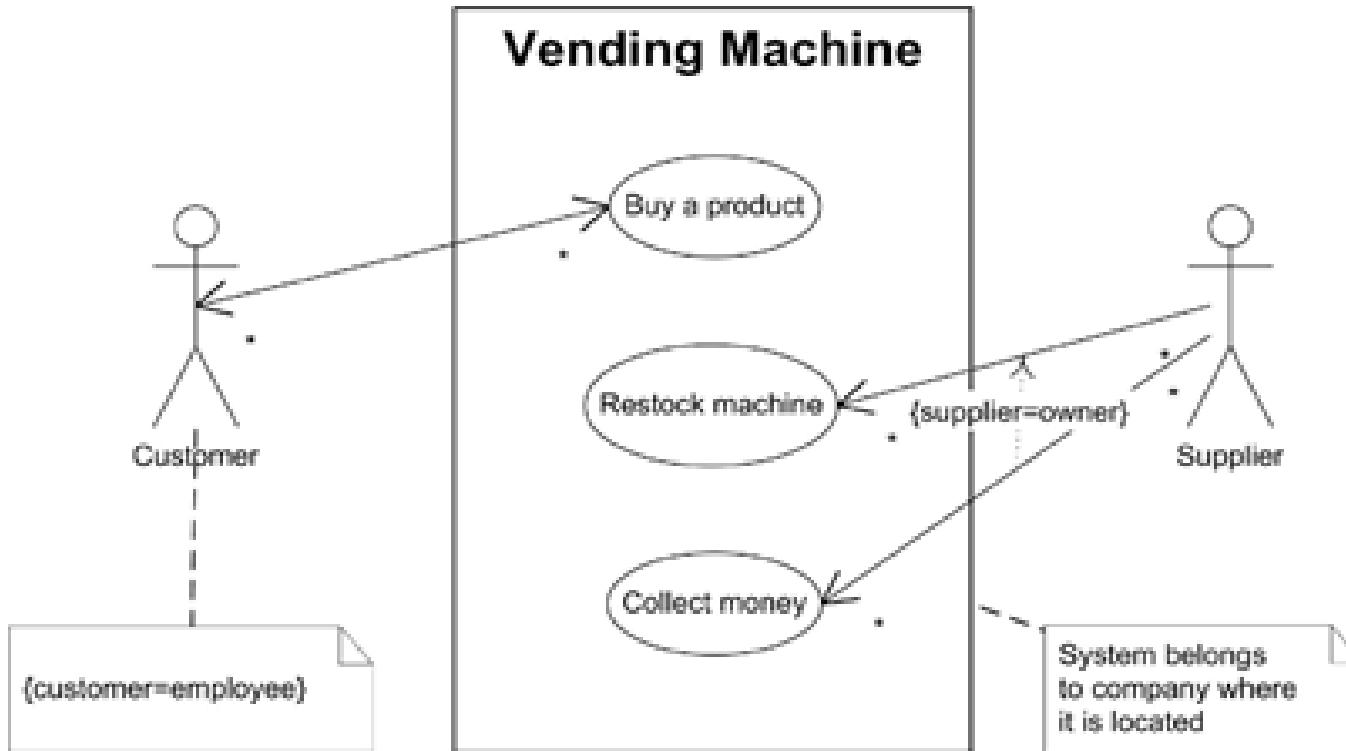
Customer; Supplier; Collector (in this case Collector=Supplier)

# Use-Case Diagram Case Study [2]



# Use-Case Diagram Case Study [3]

Introducing annotations (notes) and constraints.



# Exercises on Use Cases

Wishnu Prasetya (wishnu@cs.uu.nl)

2010/11

1. Many people have cats. Suppose you want to develop software enabling them to keep track the location of their cats; give a *use case diagram* modeling this software. In this example, just decide by yourself what you want from this software.

Theory questions:

- (a) What is the main purpose of use case modeling?

**Answer:** To capture the software's functional requirement.

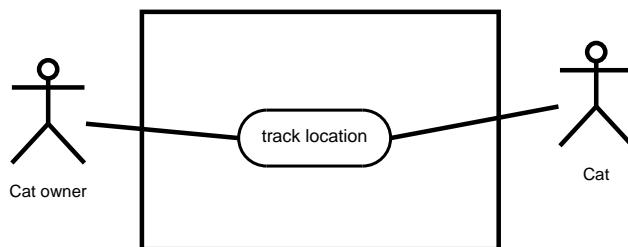
- (b) Give few examples of non-functional requirements that could be sensible for this software. How do you position use-case modeling with respect to such requirements?

**Answer:** E.g. that the software should be platform independent (can run on Windows, Mac, Unix, etc); and that it should be web-browser-based. Such requirements cannot be captured with a use case and should thus be documented complementary.

Modeling question: should 'cat' be an actor?

**Answer:** Either way can. You can see 'cat' as a (secondary) actor that provides its location to your software (e.g. through some small hardware attached to it). Or you can see it as a part of the system.

**Answer:**



Just a minimalistic one.

2. Work out at least one use case in your model above to the *casual* detail level.

Work out your use-case further to the *fully dressed* level; while you do that, give an example of an alternate flow that would be sensible in your use-case.

Should you just keep one model, or both models?

**Answer:**

Casual:

---

**Use case:** track location.

**Primary actor:** Owner

**Secondary actors:** Cat

The owner requests the locations of her cat(s). The system shows her their last known locations.

In order to know the locations, the system regularly checks the cats.

---

Fully dressed:

---

**Use case:** track location.

**Primary actor:** Owner

**Secondary actors:** Cat

**Pre-condition:** -

**Post-condition:** -

**Main flow:**

A1. The use-case is activated when the Owner requests it.

A2. The system checks the last known locations of the owner's cats.

A3. The system displays the location.

To keep track the locations, the system continuously polls the cats in the background:

B1. loop forever:

B2. for each cat, check the cat location. Update knowledge with this location.

**Alternate flow:**

A3b. If the last known location of a cat is timed has not changed since the last time the owner asked, and it is too old (> 3 hour), warn the owner.

---

3. Give situations (one for each) where it would be sensible to use these constructs to structure your use case above:

- << include >>
- << extend >>
- specialization

**Answer:**

(a) E.g. if the polling in the example above is quite involved, we can separate it to its own use case, and link it with << include >>.

(b) If the alternate flow in the example above becomes too large, we can separate it in its own use case, and link it with << extend >>.

(c) Suppose we also provide tracking of wild animals, which could be a bit different than tracking a house cat. E.g. we need to poll less frequently. A use case for this can be made as a *specialization* from that of tracking cat.

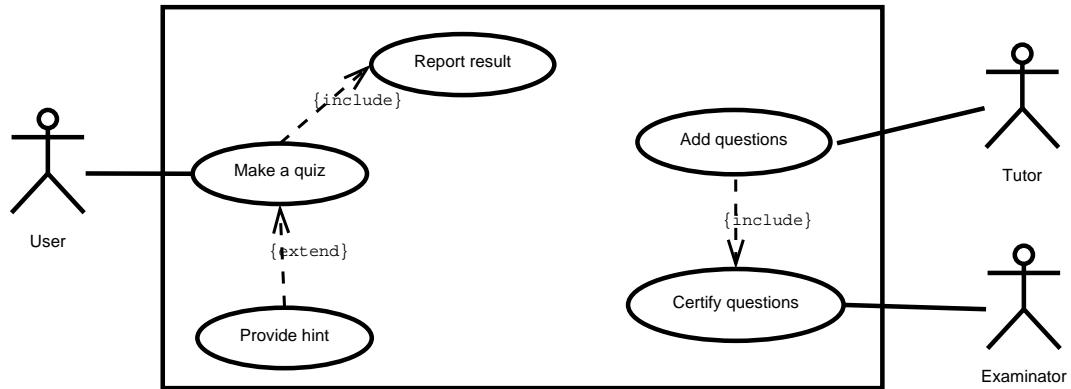
4. Do you know that it costs a lot of money to get a 'Certified Java Programmer' certificate? It could cost you thousands of euros. Let's imagine we will develop a browser-based training system to help people prepare for such a certification exam.

A user can request a quiz for the system. The system picks a set of questions from its database, and compose them together to make a quiz. It rates the user's answers, and gives hints if the user requests it.

In addition to users, we also have tutors who provide questions and hints. And also examiners who must certify questions to make sure they are not too trivial, and that they are sensible.

Make a use case diagram to model this system. Work out some of your use cases. Since we don't have real stakeholders here, you are free to fill in details you think is sensible for this example.

**Answer:**



We'll assume multiple choice quiz.

**Use case:** Make quiz.

**Primary actor:** User

**Secondary actors:** -

**Pre-condition:** The system has at least 10 questions.

**Post-condition:** -

**Main flow:**

1. The use-case is activated when the user requests it.
2. The user specifies the difficulty level.
3. The system selects 10 questions, and offers them as a quiz to the user.
4. The system starts a timer.
5. For every question:
  - 5a. The user selects an answer, or skip. [Extension point]
6. If the user is done with the quiz, or the timer runs out, the quiz is concluded, and [include use case 'Report result'].

**Use case:** Provide hint

**Primary actor:** User

**Secondary actors:** -

**Pre-condition:** The user requests for a hint.

**Post-condition:** -

**Main flow:**

1. The system provides a hint. The verbosity of the hint is determined by the difficulty level set previously by the user.
2. Return to 'Make quiz' main flow.

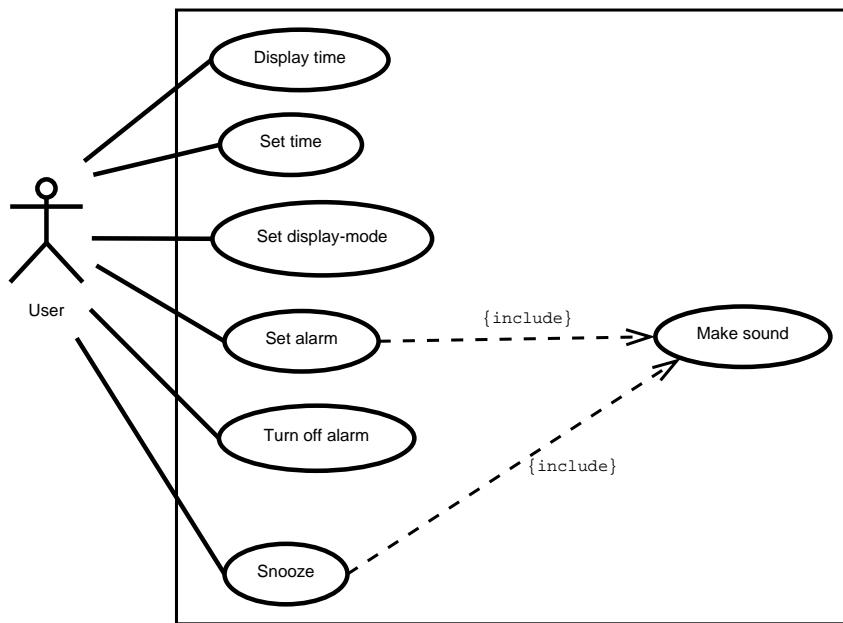
5. Suppose we want to develop software for an alarm clock.

The clock shows the time of day. Using buttons, the user can set the hours and minutes fields individually, and choose between 12 and 24-hour display.

It is possible to set one or two alarms. When an alarm fires, it will sound some noise. The user can turn it off, or choose to 'snooze'. If the user does not respond at all, the alarm will turn off itself after 2 minutes. 'Snoozing' means to turn off the sound, but the alarm will fire again after some minutes of delay. This 'snoozing time' is pre-adjustable.

- Identify the top-level functional requirement for the clock, and model it with a use case diagram.

**Answer:**



In this model 'make sound' is made as a separate use case. It does not have to.

- I assume that 'snooze' would be one of your use cases. Work it out to the fully dressed level.

**Answer:**

---

**Use case:** Snooze.

**Primary actor:** User

**Secondary actors:** -

**Pre-condition:** An alarm is firing.

**Post-condition:** -

**Main flow:**

1. The use-case is activated when the user hits the snooze button.
  2. The alarm is turned off.
  3. Wait for snooze time.
  4. Include use case 'Make sound'
- 

---

**Use case:** Make sound

**Primary actor:** System

**Secondary actors:** -

**Pre-condition:** -

**Post-condition:** -

**Main flow:**

The use case starts when it is called. What it does is to just make some noisy sound.

---

# Sample Exam Solutions

Name (Capital letters)

Surname (Capital letters)

Matriculation Number



Universität  
Zürich<sup>UZH</sup>

## Final Exam Requirements Engineering I (MINF 4204, HS13)

4th November 2013

- You have **90 Minutes** time to solve the final exam. In total, you can accumulate maximum **90 Points**.

**The following rules apply to this written final exam:**

- Answer the questions directly on the exam sheets. Indicate clearly which answer corresponds to which question. Additional sheets of paper will be distributed upon request. If you use additional sheets, write your matriculation number as well as your name and surname on each individual additional sheet.
- Please check if your exam is complete (12 pages).
- Write your solutions with a pen in **blue** or **black color**. Pens or pencils in other colors are not allowed. The solutions which are written in pencil will not be corrected.
- Always use the **notation** introduced in the **lecture**.
- For the Requirements Engineering I exam, only the following **helping materials** are allowed:
  - One double-sided self-written A4 auxiliary sheet, with handwritten notes. The auxiliary sheets which do not conform to these guidelines will be collected.
  - For students, whose native language is not English: a foreign dictionary. This will be checked by one exam supervisor.
- No additional tools or materials are allowed, particularly pocket calculators, computers, smartphones, audio devices or similar. Any fraud attempt leads to failing the exam (i.e. 0 points).
- Place your student ID („Legi“) on your desk.

I certify with my signature that I have **read** and **understood** the guidelines and rules above.

Signature: \_\_\_\_\_

Please leave empty!

Part	1	2	3	Total
Max	20	40	30	90

## **Part 1: Multiple Choice Questions (total 20 Points, ca. 20 minutes of work)**

Please check the appropriate box. For each question, there may be multiple right and wrong statements.

Note that the number of points removed for an incorrect check is equal to the number of points granted for a correct check. If the total number of points for a question is negative, then the score for that question is 0.

### **Question 1.1: Basics (3 Points)**

	<b>Correct</b>	<b>Wrong</b>
A stakeholder is a person or organization that directly influences the requirements of the system and actively contributes to the requirements validation activity.	<input type="checkbox"/>	<input checked="" type="checkbox"/>
As long as the opinions of various stakeholders do not conflict, there is no need to use requirements engineering practices.	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Dealing with multi-level requirements represents one major challenge posed by systems-of-systems to requirements engineering.	<input checked="" type="checkbox"/>	<input type="checkbox"/>
When eliciting requirements, one has to consider those both inside and outside the system and context boundaries.	<input type="checkbox"/>	<input checked="" type="checkbox"/>
In the waterfall model, it is assumed that all the requirements can be elicited at the beginning of the project.	<input checked="" type="checkbox"/>	<input type="checkbox"/>
The effort invested in requirements engineering shall be inversely proportional to the risk one is willing to take.	<input checked="" type="checkbox"/>	<input type="checkbox"/>

### **Question 1.2: Requirements and shared understanding (2 Points)**

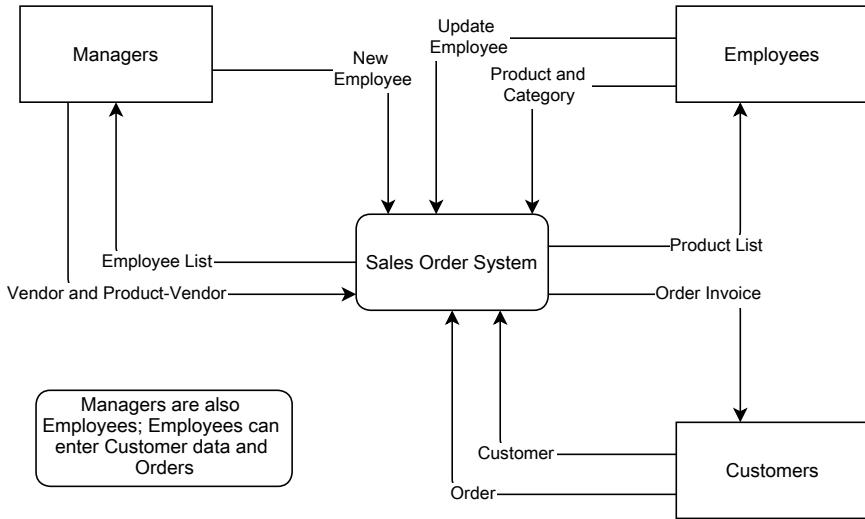
	<b>Correct</b>	<b>Wrong</b>
System requirements can be functional requirements, quality requirements or constraints.	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Soft requirements use a binary acceptance criterion for expressing the satisfaction level.	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Shared understanding refers to specifying all the requirements shared by various stakeholders.	<input type="checkbox"/>	<input checked="" type="checkbox"/>
One method to assess shared understanding is validating all explicitly specified requirements.	<input checked="" type="checkbox"/>	<input type="checkbox"/>

### **Question 1.3: Documenting requirements (2 Points)**

	<b>Correct</b>	<b>Wrong</b>
The main documentation standards include VOLERE, IEEE Std 830-1998 and enterprise-specific standards.	<input checked="" type="checkbox"/>	<input type="checkbox"/>
The reaction time and cultural issues are examples of aspects that should be documented in requirements specifications.	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Glossaries represent a method to increase the precision of the requirements documentation.	<input checked="" type="checkbox"/>	<input type="checkbox"/>
The three main directions that should be considered when writing a requirements documentation are: precision, depth and correctness.	<input type="checkbox"/>	<input checked="" type="checkbox"/>

### Question 1.4: Context diagram (2 Points)

Consider the following context diagram:



Correct	Wrong
---------	-------

The context diagram is incomplete since it does not show where the data reside and how they flow.

The context diagram is incomplete because it is missing the cardinalities.

The context diagram is at the right level of detail, such that all the stakeholders can understand it.

The managers, employees and customers can also be modelled using other symbols, other than boxes.

### Question 1.5: Models (3 Points)

Correct	Wrong
---------	-------

The Entity Relationship Model (ERM) models an extract of reality with the help of entities, relationships, methods and attributes.

Data flow diagrams have the advantage that they support system decomposition.

Class and object diagrams model the static structure of a system, together with the behavior of individual classes or objects.

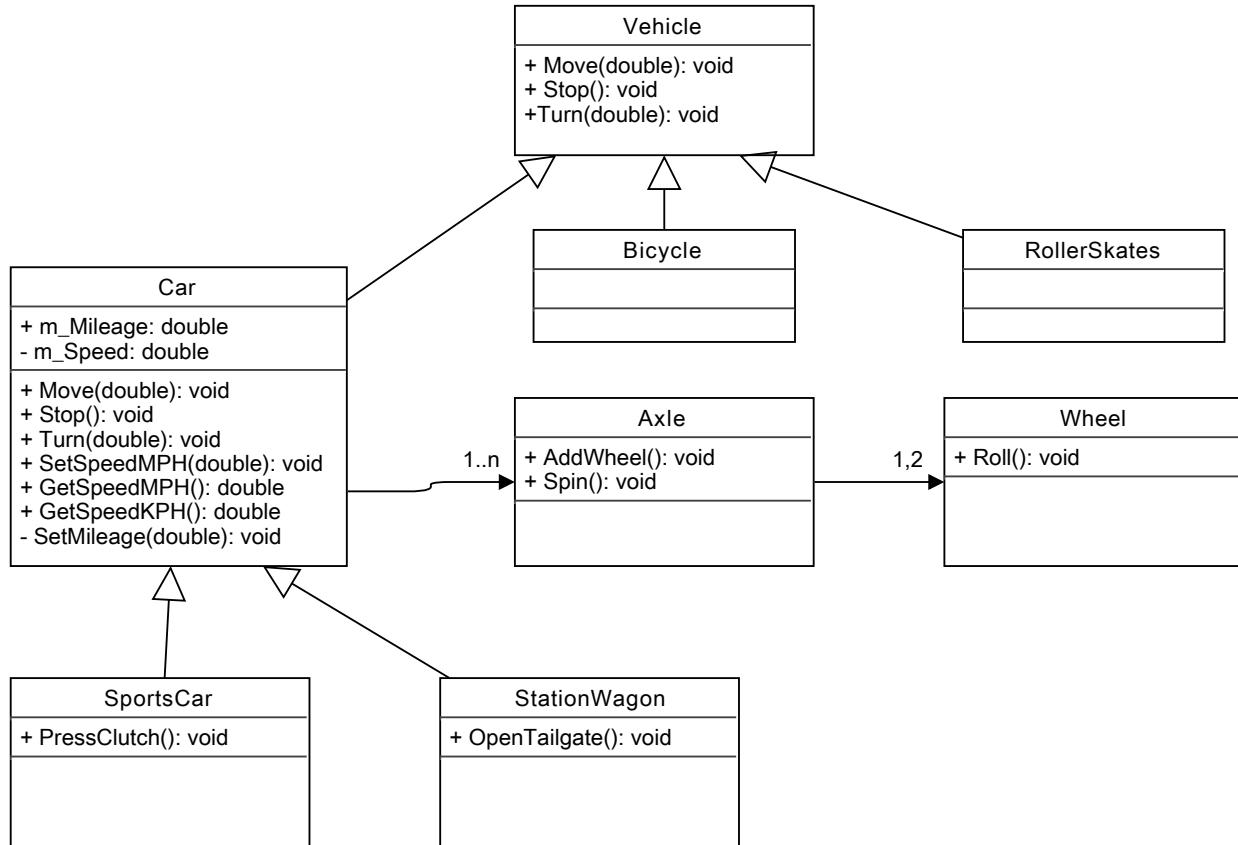
In addition to the static structure of the system, an entity relationship diagram also models some parts of the system behavior.

The behavior of a system can be modeled with an activity diagram or a state machine.

Scenarios can be represented with class diagrams.

### Question 1.6: Class diagram (2 Points)

Consider the following class diagram:



Correct	Wrong
---------	-------

The class **Vehicle** does not contain any attributes.

The objects of class **StationWagon** do not have any attributes.

The diagram contains a mistake: the direction of the five arrows with empty arrowheads pointing at **Vehicle** and **Car** should be reverse.

**SportsCar** has a method called `AddWheel()`.

### Question 1.7: Functional and quality requirements (3 Points)

Consider an online shop for books. Check if the following statements express functional or quality requirements.

Functional	Quality
------------	---------

The system shall be available in English, German and French.

The system shall allow the user to search for books by author and title.

The system shall provide a list of all previously ordered books to the user.

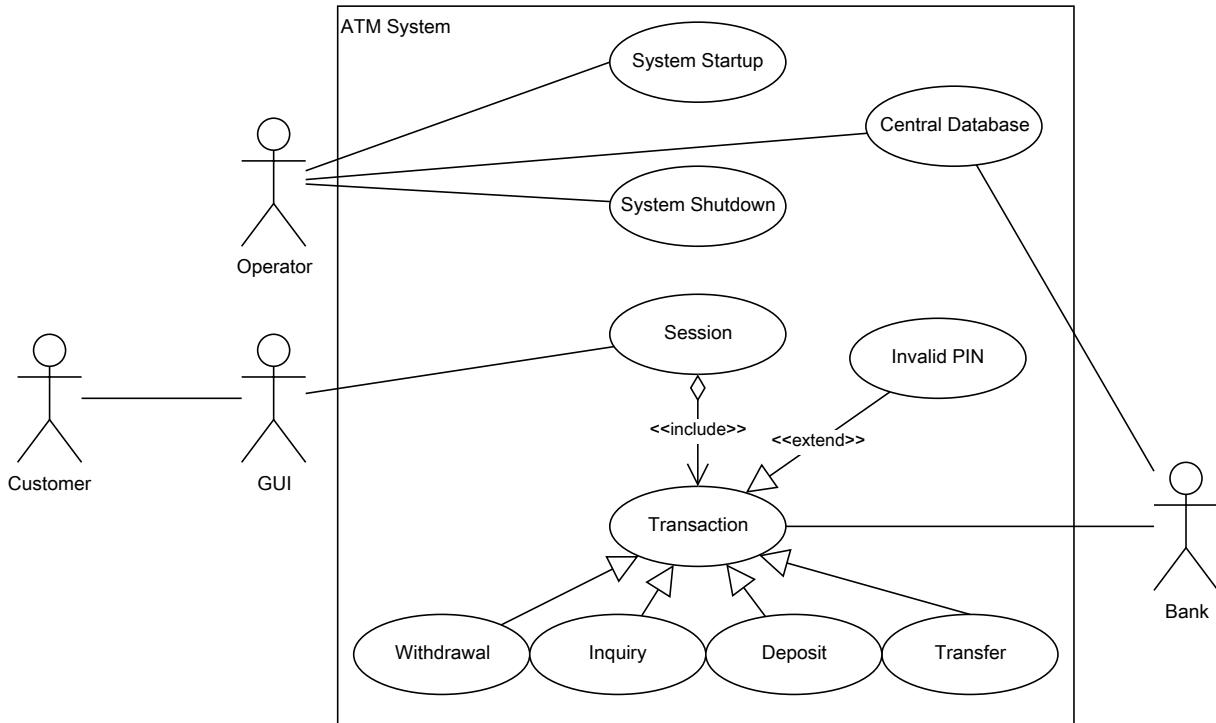
The system shall support minimum 1000 transactions per hour.

The system shall be available to all users 24/7.

The system shall allow the user to remove books from the shopping cart at any moment.

### Question 1.8: Use case diagram (3 Points)

Consider the following use case diagram:



Correct	Wrong
<input type="checkbox"/>	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input checked="" type="checkbox"/>
<input type="checkbox"/>	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input checked="" type="checkbox"/>

According to the diagram, Transaction is an abstract superclass for Withdrawal, Inquiry, Deposit, Transfer and Invalid PIN.

The relation between Invalid PIN and Transaction does not conform to the UML standard.

The use case should clarify in what direction data is transferred to and from the Central Database.

The Central Database should be moved outside the ATM System box, but the connections should be kept.

The relation between the Customer and the GUI is not permitted in UML use case diagram syntax.

The relations connecting the Operator, GUI and Bank to the ATM System are missing the arrows.

## **Part 2: Applied Tasks (total 40 Points, ca. 40 minutes of work)**

### **Case Study**

JazzN!ghts is a famous Jazz festival, held in Zurich every year. Since its first edition in 1986, it has gone through several major changes regarding its structure, length and location, but the tickets have always been sold in a traditional way: through two events agencies. The organizers decided to completely modernize the tickets selling system and created the following concept.

From this year on, the tickets will be sold in three distinct ways: traditionally, i.e. by the two events agencies, in electronic format directly on the festival website, and through SBB. All parties will have access to the same unique tickets database of the new system, to avoid double selling. A partnership with the SBB railway company needs to be set up, such that SBB can sell combi-tickets including both the festival admission fee and the train ride to the festival venue at reduced price, from anywhere in Switzerland. This way, more music fans would have easier and cheaper access to JazzN!ghts. Moreover, the system will have to be extended to support not only German, but also English, French and Italian. Since tickets will also be sold online, SecurePayment Inc. will be contracted to provide and ensure the security of the online payment service. The JazzN!ghts event manager will take care and negotiate all these details with the involved parties.

Additionally, upon arrival at the festival venue, each participant has to self-check in at a touch screen terminal, which scans the barcodes on his/her ticket and issues a bracelet with an electronic chip. This can be used to load money, such that whenever (s)he wants to purchase snacks or beverages, (s)he does not have to use cash any more, thus reducing waiting times. This measure was initiated by the program manager and will be deployed by WristSolutions Inc. Lastly, according to the cantonal laws, the way the payment transactions are performed has to be audited by an external company at the end of the festival, since this is a public event, where the municipality of Zurich is also involved - allowing free use of the public space.

- a. Identify and name all the stakeholders of the JazzN!ghts new tickets selling system. **(4 Points)**

### **Musterlösung**

List of stakeholders:

- 1) The organizers of JazzN!ghts
- 2) The event manager of JazzN!ghts
- 3) The customers
- 4) Two events agencies
- 5) SBB
- 6) SecurePayment Inc.
- 7) WristSolutions Inc.
- 8) External auditing company
- 9) Municipality of Zurich

#### **Correction schema:**

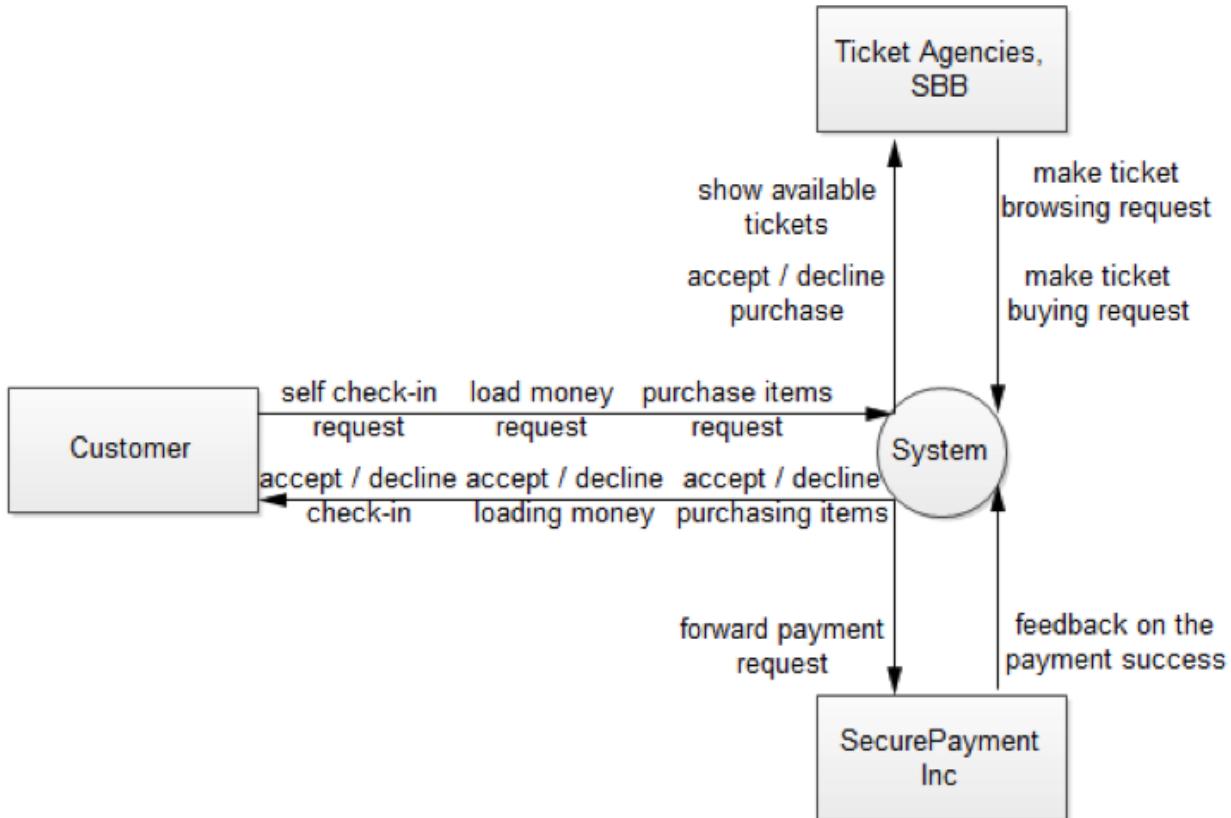
Students are expected to list at least 8 of the stakeholders above.

0.5 points/stakeholder \* 8 stakeholders => 4 points

- b. Draw a context diagram of the new JazzN!ghts tickets selling system. Make sure you label the actors, the relationships between them as well as their relationships to the system. Note: Do not forget to document your assumptions, if you make any. (10 Points)

### Musterlösung

Note: This is a sample solution, but other versions that include the same actors and relationships are also acceptable. More than one solution is possible.



#### **Correction schema:**

1 point/actor \* 3 actors + system => 4 points

1 point/relationship \* 6 relationships => 6 points

Total: 10 points

- c. Identify one goal, two functional requirements and two non-functional requirements in the case study. (5 Points)

### Musterlösung

Sample **goal**: Modernize the tickets selling system.

Sample **functional requirements**:

- 1) The system enables purchasing tickets;
- 2) The system allows performing a self-check-in by using the barcode of the ticket;
- 3) The system allows loading money on the bracelet;
- 4) The system allows purchasing items on the festival venue using the money loaded on the bracelet.

Sample **non-functional requirements**:

- 1) The system is available in German, English, French and Italian;
- 2) The online payment service is secure.

#### **Correction schema:**

1 point/goal

1 point/functional requirement \* 2 functional requirements => 2 points

1 point/non-functional requirements \* 2 non-functional requirements => 2 points

Total: 5 points

- d. How can you gather further requirements that help you build the JazzN!ghts new tickets selling system? Mention *four* different requirements elicitation methods. For each, explain (i) why you think the method is suitable in the context of the JazzN!ghts case study, (ii) what stakeholders the method is appropriate for, and (iii) how you would implement the method in practice, for JazzN!ghts. **(12 Points)**

## **Musterlösung**

**Note:** More than one answer is possible. The solution below is an example.

### **Method 1:** Questionnaires

- i) Why: Questionnaires can reach a large and distributed audience the system is aimed for, such as the Jazz fans.
- ii) Stakeholders: E.g. (potential) customers, i.e. Jazz fans.
- iii) How: The link to an online questionnaire can be sent by e-mail to Jazz fans in Switzerland - the data can be retrieved from e.g. Jazz clubs.

### **Method 2:** Interview

- i) Why: During an interview with (potential) users, detailed features can be analyzed and the system designer can gain an in-depth understanding of individual needs.
- ii) Stakeholders: Sample future users, i.e. Jazz fans,
- iii) How: A few Jazz fans volunteers can be contacted and selected through existing Jazz clubs, then invited to take part in structured or semi-structured interviews regarding the new tickets system.

### **Method 3:** Workshop

- i) Why: A workshop with the external auditing company would clarify what their needs are, what they need to audit and what kind of information the system should provide.
- ii) Stakeholders: External auditing company.
- iii) How: Audit company representatives can be invited to participate in the workshop and simulate what an auditing session would look like.

### **Method 4:** Ethnography

- i) Why: In this case, ethnography would allow observing users while they use a prototype of the future system.
- ii) Stakeholders: E.g. (potential) customers, i.e. Jazz fans.
- iii) How: Selected future users can be invited to try a prototype of the system. During this time, requirements specialists can observe them, note down their interactions with the system, potential issues, etc.

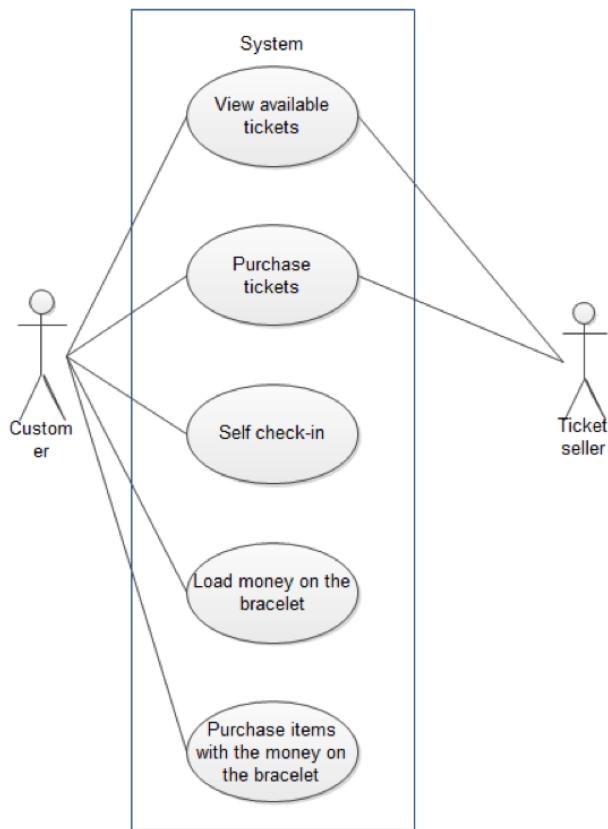
### **Correction schema:**

3 points/method \* 4 methods => 12 points (1 point for (i), 1 point for (ii), and 1 point for (iii))

- e. Identify all the use cases in the JazzN!ghts case study and represent them in a UML use case diagram. Note: Do not forget to document your assumptions, if you make any. **(9 Points)**

## **Musterlösung**

**Note:** More than one answer is possible. The solution should contain at least the use cases below.



**Correction schema:**

1 point/actor \* 2 actors => 2 points  
1 point/use case \* 5 use cases => 5 points  
2 points/relations  
Total: 9 points

### **Part 3: Essays (total 30 Points, ca. 30 minutes of work)**

Please answer each of the following questions in a short essay, of about 8-16 sentences. For this, you should use coherent text, and not bullet points.

- a. Discuss the problems related to eliciting quality requirements and explain how these can be avoided. **(10 Points)**

#### **Musterlösung**

**Sample problems** the students are expected to mention:

- quality requirements are generally ambiguous
- difficult to achieve
- difficult to verify and quantify: e.g. "The system shall be fast" or "We need a secure system"

**Classic approach:**

- quantification: measurable, but expensive
- operationalization: testable, but implies premature design decisions

**New approach:** represent quality requirements such that they deliver optimum value

**Correction schema:**

Students are expected to mention minimum two problems and corresponding possible solutions.

2 point/problem \* 2 problems => 4 points

2 point/solution \* 2 solutions => 4 points

2 points/writing

Total: 10 points

- b. Describe and discuss the requirements validation techniques, and explain in what contexts it is appropriate to use them.  
**(10 Points)**

## **Musterlösung**

Requirements validation techniques:

1. Review
  - walkthrough
  - inspection
  - author-reviewer cycle
2. Acceptance test cases
3. Simulation/Animation
4. Prototyping
5. Formal verification/Model checking

### **Correction schema:**

2 points/validation technique \* 5 techniques => 10 points (1 point for naming the technique, 0.5 points for short description of the technique, and 0.5 points for explaining in what context it is appropriate to use it)

- c. Describe and discuss the characteristics of the agile requirements process. (**10 Points**)

### **Musterlösung**

The students are expected to describe and discuss minimum 5 of the following characteristics:

- fixed length increments of 1-6 weeks
- products owner/customer available
- immediate decisions are possible
- only goals and vision established upfront
- loosely specified requirements as user stories
- details captured in test cases
- short feedback cycles
- at the beginning of each cycle, the customer/product owner prioritizes the requirements and the developers select the requirements to be implemented in that increment

#### **Correction schema:**

2 points/characteristic of the agile requirements process \* 5 characteristics => 10 points

Space for Solutions