Object Oriented Analysis and Design

By
Dr. L. Ranathunga

1

Additional References

- Visual Modeling with Rational Rose 2002 and UML – Terry Quatrani
- Rational Software Architect – Quick start guide
- Object Oriented modeling and Design – James Rambaugh, Michele Blaha, el.,

2

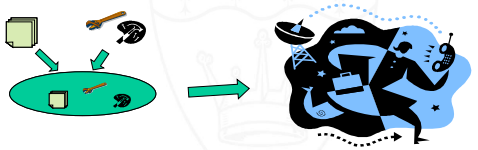
What is Object-Oriented?

- Object oriented analysis and design is a bottom-up way of thinking about problems using models organized around real-world concepts.
- The fundamental building block is the object.
- Object combines data structures and behaviors in a single entity.

3

What is Object-Oriented?

- Organisation of software as a collection of discrete objects that incorporate both data structures and behaviors.



- This is in contrast to conventional programming in which data structures and behaviors are loosely connected.

4

Behavioral Models

- Systems have static & dynamic characteristics
  - Structural models describe the static aspects of the system
  - Behavioral models describe the dynamics and interactions of the system and its components
- Behavioral models describe how the classes described in the structural models interact in support of the ***use cases***.

5

Characteristics of Objects

- Identity
  - The data is quantized into discrete, distinguishable entities called objects.
  - ex : wheel of a bike, paragraph of a text, window on work station, etc
- Classification
  - The objects with same data structure (attributes) and behaviors (methods) are grouped into a *class*.
  - Ex: Bingo is an object, belongs to dog class and animal super class (Abstract)

6

Characteristics of Objects

- Polymorphism
  - The same behavior may occur differently in different classes.
  - ex : move operation on a window is differ from move operation on file, etc
- Inheritance
  - the sharing of attributes and operations among classes based on hierarchical relationship.
  - ex: classes can create sub classes

7

OO Design & Development

- Problems are changing but problem-domain remains
- Customer requirements may change but business environment is same
- OOAD address problem-domain
- Structured methods address problem
- Reusability, Modularity, portability, Modifiability are there

8

What is Object-Oriented Development?

- New way of thinking about software based on abstraction that exist in real world
- In this context development refers to the front portion of the software life cycle: analysis, design, Implementation
- The essence of OO development is the identification and organisation of application-domain concepts, rather than their final representation in a programming language, OO or not

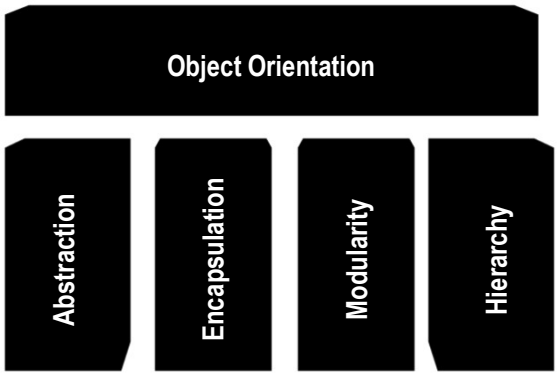
9

What is Object-Oriented Development?

- OO development approach encourages software developers to work and think in terms of the application domain through most of the SE life cycles.
- It is independent of a programming language until final stage.
- It uses for communication & specification description tools, documentation, interfacing, programming

10

Basic Principles of Object Orientation



11

What is Abstraction?

Salesperson

Not saying Which salesperson – just a salesperson in general!!!

Customer

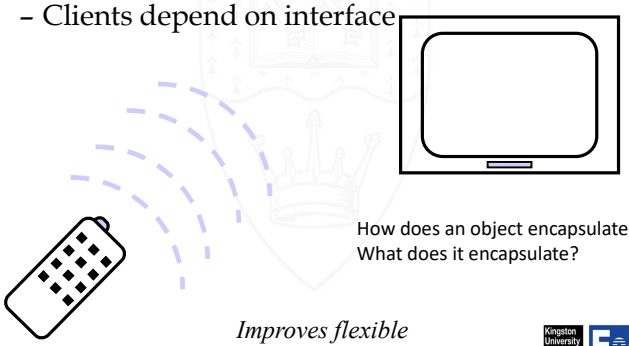
Product

Manages Complexity

12

What is Encapsulation?

- Hide implementation from clients
  - Clients depend on interface



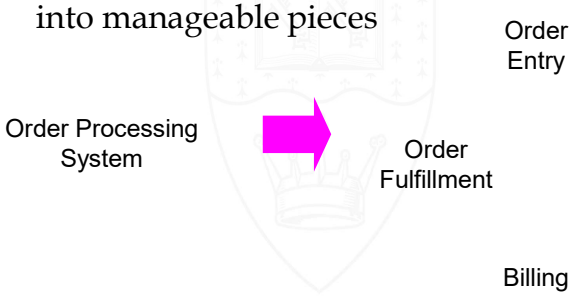
How does an object encapsulate?
What does it encapsulate?

Improves flexible

13

What is Modularity?

- The breaking up of something complex into manageable pieces

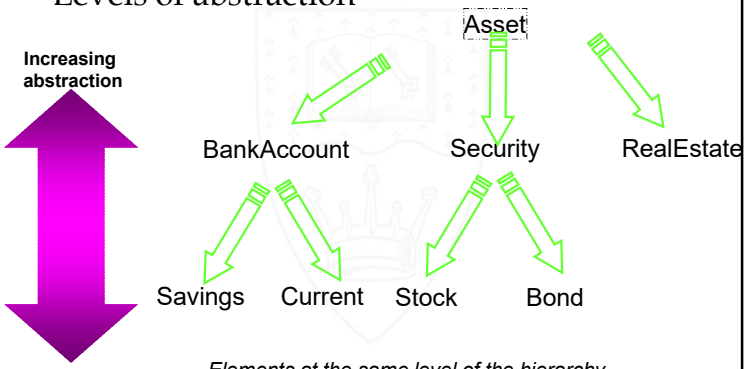


Manages Complexity

14

What is Hierarchy?

- Levels of abstraction



Elements at the same level of the hierarchy should be at the same level of abstraction

15

Basic Concepts of Object Orientation

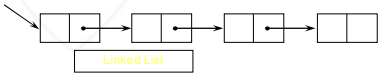
- Object
- Class
- Attribute
- Operation
- Interface (Polymorphism)
- Component
- Package
- Subsystem
- Relationships

16

What is an Object?

- Informally, an object represents an entity, either physical, conceptual, or software
  - Physical entity

  - Conceptual entity



  - Software entity

17

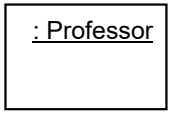
A More Formal Definition

- An object is a concept, abstraction, or thing with sharp boundaries and meaning for an application
- An object is something that has:
  - State
  - Behavior
  - Identity

18

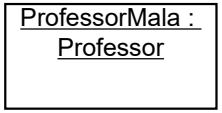
Representing Objects

- An object is represented as rectangles with underlined names



Class Name Only

Object Name Only



Class and Object Name

19

What is a Class?

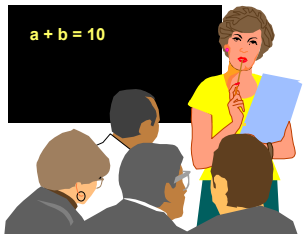
- A class is a description of a group of objects with common properties (attributes), behavior (operations), relationships, and semantics
  - An object is an instance of a class
- A class is an abstraction in that it:
  - Emphasizes relevant characteristics
  - suppresses other characteristics

20

Sample Class

Class
Course

Properties
Name
Location
Days offered
Credit hours
Start time
End time

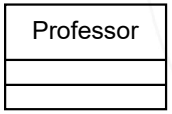


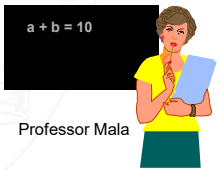
Behavior
Add a student
Delete a student
Get course time table
Determine if it is full

21

Representing Classes

- A class is represented using a compartmented rectangle

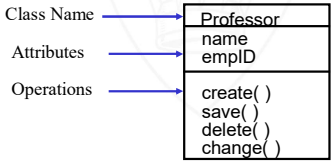




22

Class Compartments

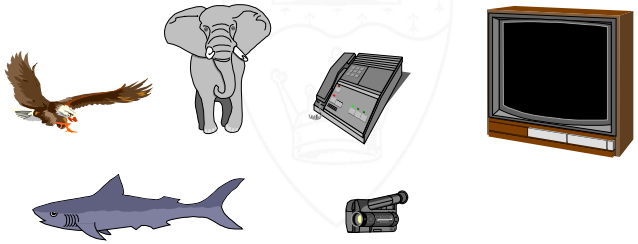
- A class is comprised of three sections
  - The first section contains the class name
  - The second section shows the structure (attributes)
  - The third section shows the behavior (operations)



23

Classes of Objects

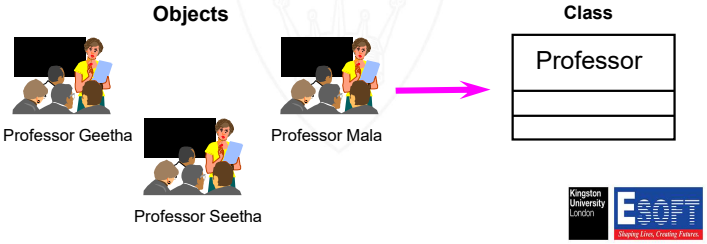
- How many classes do you see?



24

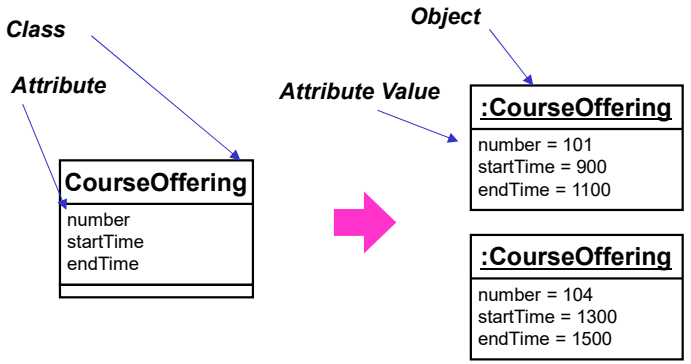
The Relationship Between Classes and Objects

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



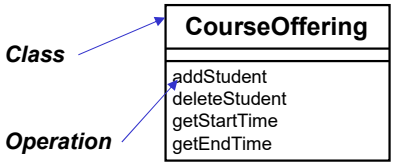
25

What is an Attribute?



26

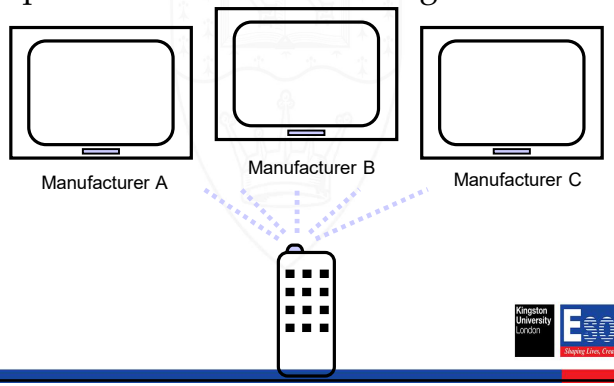
What is an Operation?



27

What is Polymorphism?

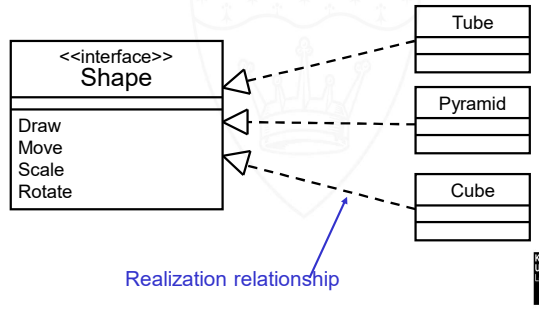
- The ability to hide many different implementations behind a single interface



28

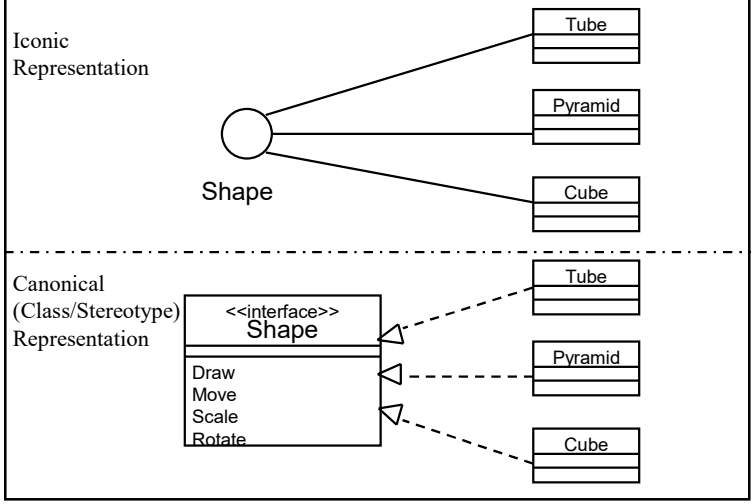
What is an Interface?

- Interfaces formalize polymorphism
- Interfaces support "plug-and-play" architectures



29

Interface Representations



30

What is a Component?

- A non-trivial, nearly independent, and replaceable part of a system that fulfills a clear function in the context of a well-defined architecture
- A component may be
  - A source code component
  - A run time components or
  - An executable component

OO Principle: Encapsulation

31

What is a Package?

- A package is a general purpose mechanism for organizing elements into groups
- A model element which can contain other model elements

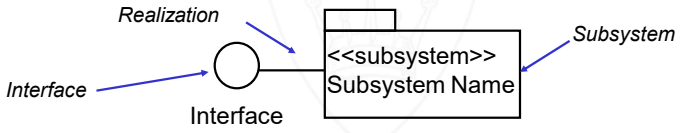
OO Principle: Modularity

- Uses
  - Organize the model under development
  - A unit of configuration management

32

What is a Subsystem?

- A combination of a package (can contain other model elements) **and** a class (has behavior)
- Realizes one or more interfaces which define its behavior

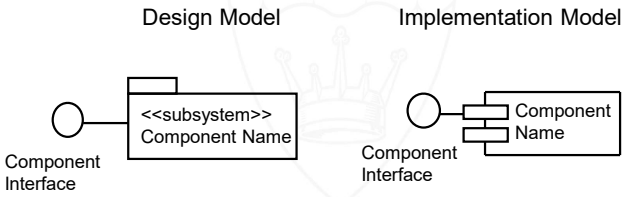


OO Principles: Encapsulation and Modularity

33

Subsystems and Components

- Components are the physical realization of an abstraction in the design
- Subsystems can be used to represent the component in the design

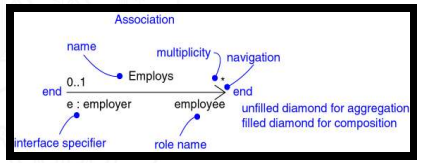


OO Principles: Encapsulation and Modularity

34

Relationships

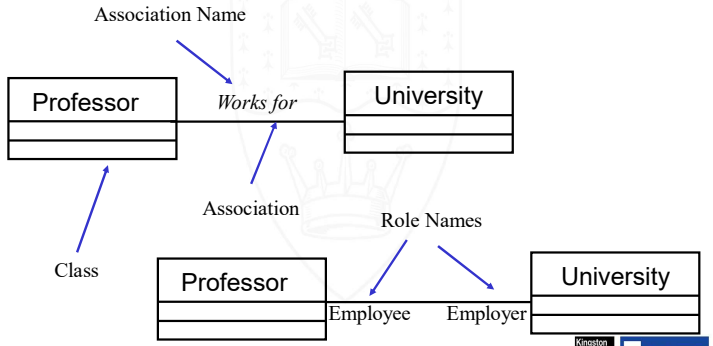
- Association
- Aggregation
- Composition
- Dependency
- Generalization
- Realization



35

Relationships: Association

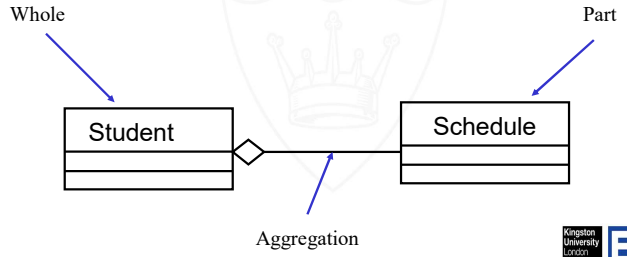
- Models a semantic connection among classes



36

Relationships: Aggregation

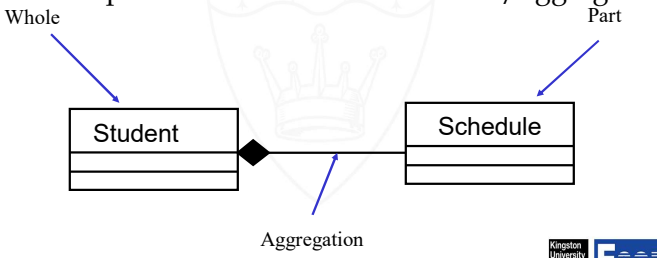
- A special form of association that models a whole-part relationship between an aggregate (the whole) and its parts



37

Relationships: Composition

- A form of aggregation with strong ownership and coincident lifetimes
  - The parts cannot survive the whole/aggregate



38

Association: Multiplicity and Navigation

- Multiplicity defines how many objects participate in a relationships
  - The number of instances of one class related to ONE instance of the other class
  - Specified for each end of the association
- Associations and aggregations are bi-directional by default, but it is often desirable to restrict navigation to one direction
  - If navigation is restricted, an arrowhead is added to indicate the direction of the navigation

39

Association: Multiplicity

- Unspecified
- Exactly one
- Zero or more (many, unlimited)

- One or more
- Zero or one
- Specified range
- Multiple, disjoint ranges

1

0..*

*

1..*

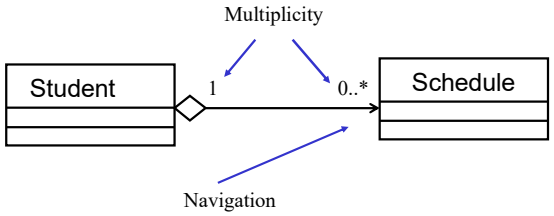
0..1

2..4

2, 4..6

40

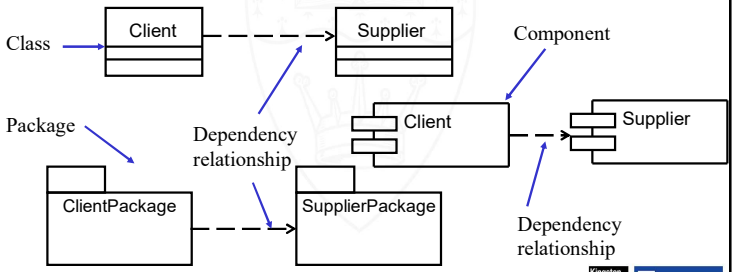
Example: Multiplicity and Navigation



41

Relationships: Dependency

- A relationship between two model elements where a change in one **may** cause a change in the other



42

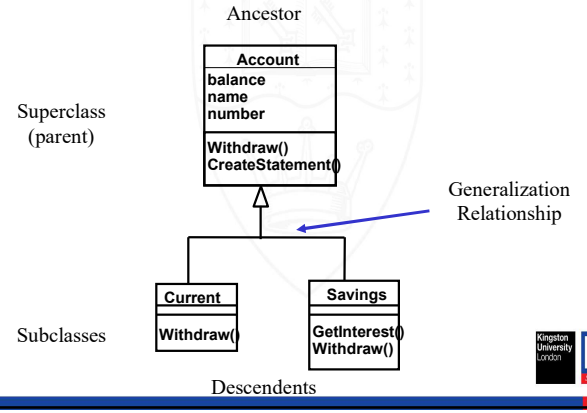
Relationships: Generalization

- A relationship among classes where one class shares the structure and/or behavior of one or more classes
- Defines a hierarchy of abstractions in which a subclass inherits from one or more superclasses
  - Single inheritance
  - Multiple inheritance
- Generalization is an "is-a-kind of" relationship

43

Example: Single Inheritance

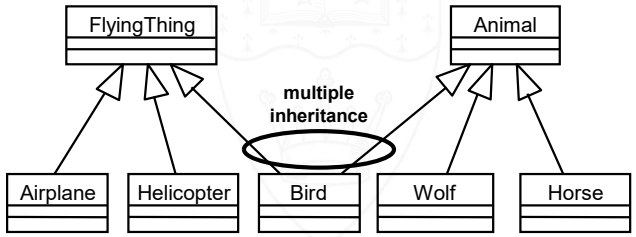
- One class inherits from another



44

Example: Multiple Inheritance

- A class can inherit from several other classes



Use multiple inheritance only when needed, and always with caution !

45

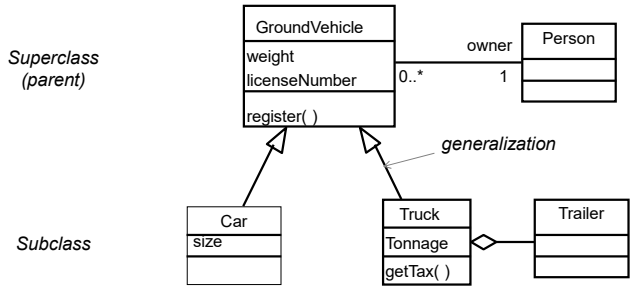
What Gets Inherited?

- A subclass inherits its parent's attributes, operations, and relationships
- A subclass may:
  - Add additional attributes, operations, relationships
  - Redefine inherited operations
- Common attributes, operations, and/or relationships are shown at the highest applicable level in the hierarchy

Inheritance leverages the similarities among classes

46

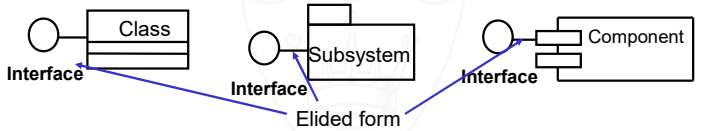
Example: What Gets Inherited



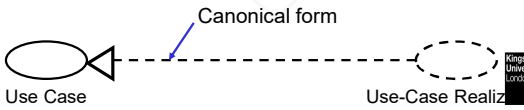
47

Relationships: Realization

- One classifier serves as the contract that the other classifier agrees to carry out
- Found between:
  - Interfaces and the classifiers that realize them



  - Use cases and the collaborations that realize them



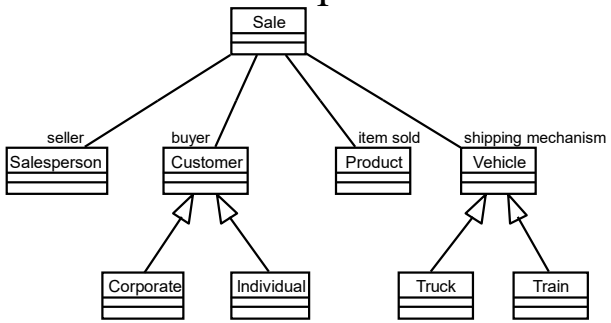
48

Strengths of Object Orientation

- A single paradigm
- Facilitates architectural and code reuse
- Models more closely reflect the real world
  - More accurately describe corporate data and processes
  - Decomposed based on natural partitioning
  - Easier to understand and maintain
- Stability
  - A small change in requirements does not mean massive changes in the system under development

49

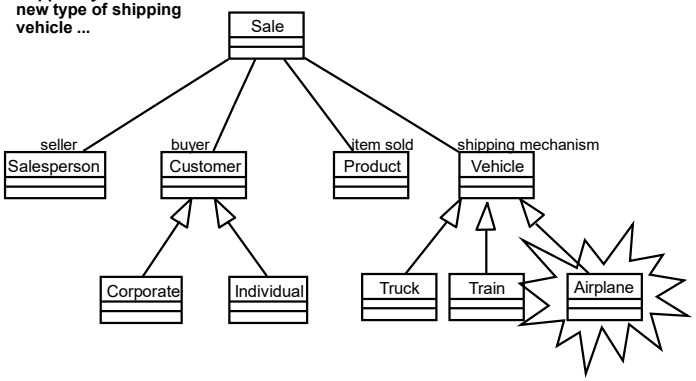
Class Diagram for the Sales Example



50

Effect of Requirements Change

Suppose you need a new type of shipping vehicle ...



Change involves adding a new subclass

51

Traditional Expressions of Functional Requirements

- Requirements specifications
  - Hard to read
- Context Diagram
  - Specifies users, software, hardware that interface with system
- Data-flow Diagrams (DFD)
  - Useful for technical people but tend to confuse users
  - Useful in design of non-object-oriented systems
- Entity-relationship diagrams (ERD)
  - Critical to database design but are not easily understood by users
- Prototypes
  - Good communication tool to elicit information from user.
  - Great for proof-of-concept tasks.
  - Useful in developing user interface designs.

52

UML Diagrams

- Instead of the Context, Data-Flow and Entity-Relationship Diagrams used in Structured Analysis, UML produces 9 types of diagrams
  - Use Case Diagram
  - Sequence Diagram
  - Collaboration Diagram
  - State chart Diagram
  - Activity Diagram
  - Class Diagram
  - Object Diagram
  - Component Diagram
  - Deployment Diagram

53

Use Cases

- **Use cases** are a means of expressing user requirements.
- Use cases are used extensively in the analysis phase.
- A **use case** represents how a system interacts with its environment by illustrating the activities that are performed by the users and the system's responses.
- The text-based use case is easy for the users to understand, and also flows easily into the creation of process models and the data model.

54

Use Cases

- A ***use case*** depicts a set of activities that produce some output result.
- Each use case describes how an external user ***triggers*** an ***event*** to which the system must respond.
- With this type of ***event-driven modeling***, everything in the system can be thought of as a response to some triggering event.
- Creation of use cases is often done as a part of interview session with users or a part of JAD sessions.

55

Scenario

- A **scenario** is a sequence of steps describing an interaction between a user and a system.

56

Buy a Product

- The customer browses the catalog and adds desired items to the shopping basket. When the customer wishes to pay, the customer describes the shipping and credit card information and confirms the sale. The system checks the authorization on the credit card and confirms the sale both immediately and with a follow-up email.

57

Use case

- A **use case**, then, is a set of scenarios tied together by a common user goal.
- Jacobson is a scholar who has written about capturing system requirements in packages of transactions called *use cases*.

58

What is a Use Case?

- The Use Cases describe the behavior of a system from a *user's standpoint* using actions and reactions.
- The Use Case Diagram defines the system's boundary, and the relationships between the system and the environment:
  - different **human users** roles interact with our system
  - other **software** systems/applications
  - **hardware** systems/devices
- Use Cases support the specification phase by providing a means of capturing and documenting requirements

59

Buy a Product

1. Customer browses through catalog and selects items to buy
2. Customer goes to check out
3. Customer fills in shipping information (address; next-day or 3-day delivery)
4. System presents full pricing information, including shipping
5. Customer fills in credit card information
6. System authorizes purchase
7. System confirms sale immediately
8. System sends confirming email to customer

Alternative: *Authorization Failure*
At step 6, system fails to authorize credit purchase
Allow customer to re-enter credit card information and re-try

Alternative: *Regular Customer*
3a. System displays current shipping information, pricing information, and last four digits of credit card information
3b. Customer may accept or override these defaults
Return to primary scenario at step 6

60

Use Case Deliverables

- There are two parts to document a use case:
  - the **use case diagram**,
    - provides visual overview of important interactions
    - captures scope (identifies external entities)
  - the **use case** itself
    - documents in a textual form the details of the requirements, what the use case must do.
    - A use case is actually a page or two of text representing each oval in the use case diagram
    - A project should have a standard template for use cases.

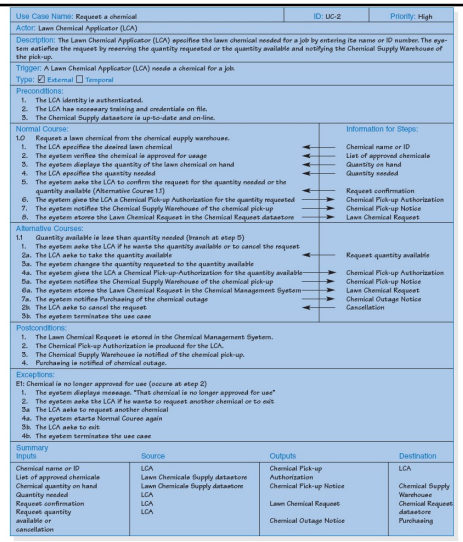
61

Elements of a Use Case

Basic Information

- Each use case has a *name* and *number*, and brief description.
- The *priority* may be assigned to indicate the relative significance.
- The *actor* refers to a person, another system, or a hardware device that interacts with the system to achieve a useful goal.
- The *trigger* for the use case – the event that causes the use case to begin.

62



Use Case Name: Request a chemical | ID: UC-2 | Priority: High

Actor: Lawn Chemical Applicator (LCA)

Description: The Lawn Chemical Applicator (LCA) specifies the lawn chemical needed for a job by entering its name or ID number. The system satisfies the request by reserving the quantity requested or the quantity available and notifying the Chemical Supply Warehouse of the pick-up.

Trigger: A Lawn Chemical Applicator (LCA) needs a chemical for a job.

Type: ☑ External ☐ Temporal

Preconditions:
1. The LCA identity is authenticated.
2. The LCA has necessary training and credentials on file.
3. The Chemical Supply datastore is up-to-date and on-line.

Normal Course:

		Information for Steps:
1.0	Request a lawn chemical from the chemical supply warehouse.	
1.	The LCA specifies the desired lawn chemical	← Chemical name or ID
2.	The system verifies the chemical is approved for usage	← List of approved chemicals
3.	The system displays the quantity of the lawn chemical on hand	← Quantity on hand
4.	The LCA specifies the quantity needed	← Quantity needed
5.	The system asks the LCA to confirm the request for the quantity needed or the quantity available (Alternative Course 1.1)	← Request confirmation
6.	The system gives the LCA a Chemical Pick-up Authorization for the quantity requested	→ Chemical Pick-up Authorization
7.	The system notifies the Chemical Supply Warehouse of the chemical pick-up	→ Chemical Pick-up Notice
8.	The system stores the Lawn Chemical Request in the Chemical Request datastore	→ Lawn Chemical Request

Alternative Courses:

1.1	Quantity available is less than quantity needed (branch at step 5)	
1.	The system asks the LCA if he wants the quantity available or to cancel the request	
2a.	The LCA asks to take the quantity available	← Request quantity available
3a.	The system changes the quantity requested to the quantity available	
4a.	The system gives the LCA a Chemical Pick-up Authorization for the quantity available	→ Chemical Pick-up Authorization
5a.	The system notifies the Chemical Supply Warehouse of the chemical pick-up	→ Chemical Pick-up Notice
6a.	The system stores the Lawn Chemical Request in the Chemical Management System	→ Lawn Chemical Request
7a.	The system notifies Purchasing of the chemical outage	→ Chemical Outage Notice
2b.	The LCA asks to cancel the request	← Cancellation
3b.	The system terminates the use case	

Postconditions:
1. The Lawn Chemical Request is stored in the Chemical Management System.
2. The Chemical Pick-up Authorization is produced for the LCA.
3. The Chemical Supply Warehouse is notified of the chemical pick-up.
4. Purchasing is notified of chemical outage.

Exceptions:
E1: Chemical is no longer approved for use (occurs at step 2)
1. The system displays message: "That chemical is no longer approved for use"
2. The system asks the LCA if he wants to request another chemical or to exit
3a The LCA asks to request another chemical
4a. The system starts Normal Course again
3b. The LCA asks to exit
4b. The system terminates the use case

Summary

Inputs	Source	Outputs	Destination
Chemical name or ID	LCA	Chemical Pick-up Authorization	LCA
List of approved chemicals	Lawn Chemicals Supply datastore	Chemical Pick-up Notice	Chemical Supply Warehouse
Chemical quantity on hand	Lawn Chemicals Supply datastore	Lawn Chemical Request	Chemical Request datastore
Quantity needed	LCA	Chemical Outage Notice	Purchasing
Request confirmation	LCA		
Request quantity available or cancellation	LCA		

63

Preconditions

- It is common practice to create smaller, more focused use cases breaking the whole process down into parts.
- It is important to define clearly what needs to be accomplished before each use case begins.
- The *preconditions* define the state the system must be in before the use case commences.

64

Normal Course

- The next part of a use case is the description of the major steps that are performed to execute the response to the event, the inputs used for the steps, and the outputs produced by the steps.
- The ***normal course*** lists the steps.

65

Alternative Courses

- **Alternative courses** depict branches (alternative paths of the steps) in logic that also will lead to a successful conclusion of the use case.

66

Post conditions

- The **postconditions** section of defines the final product of the use case.
- These postconditions also serve to define the preconditions for the next use case in the series.

67

Exceptions

- A use case should describe any error conditions or **exceptions** that may occur as the use case steps are performed.
- These are not normal branches in decision logic, but are unusual occurrences or errors that could potentially be encountered and will lead to an unsuccessful result.

68

Inputs and Outputs

- The final section of the use case summarizes the set of major **inputs** and **outputs** of the use case, along with their source or destination.

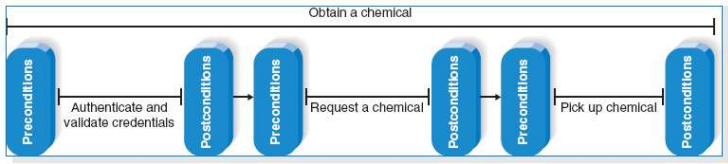
69

Additional Use Case Issues

- Additional sections may be included, e.g.,
  - Frequency of use
  - Business rules
  - Special requirements
  - Assumptions
  - Notes and issues

70

Chain of use cases – an example



71

Alternative Use Case Formats

- A *full-dressed* use case is very thorough, detailed, and highly structured.
- The project team may decide that a more casual use case format is acceptable.

72

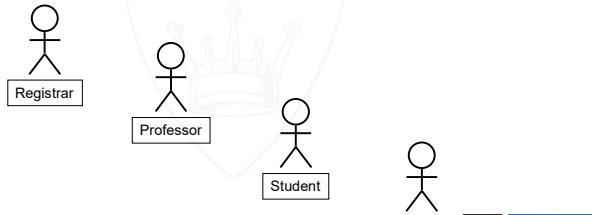
Use Cases and the Functional Requirements

- Use cases are very useful tools to us to understand user requirements. However, use cases only convey the user's point of view.
- Transforming the user's view into the developer's view by creating functional requirements is one of the important contributions of system analyst.
- The derived functional requirements give more information to the developer about what the system must do.

73

Actor

- An actor is someone or some thing that must interact with the system under development



74

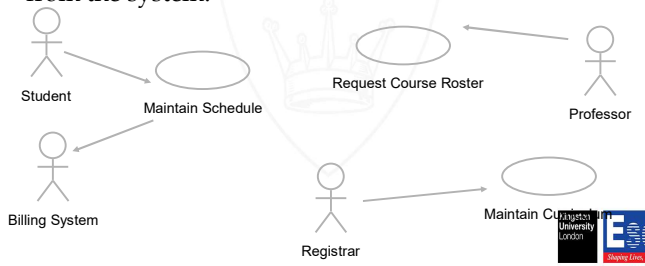
Use Cases

- A use case is a pattern of behavior the system exhibits
  - Each use case is a sequence of related transactions performed by an actor and the system in a dialogue
- Actors are examined to determine their needs
  - Registrar -- maintain the curriculum
  - Professor -- request roster
  - Student -- maintain schedule
  - Billing System -- receive billing information from registration

75

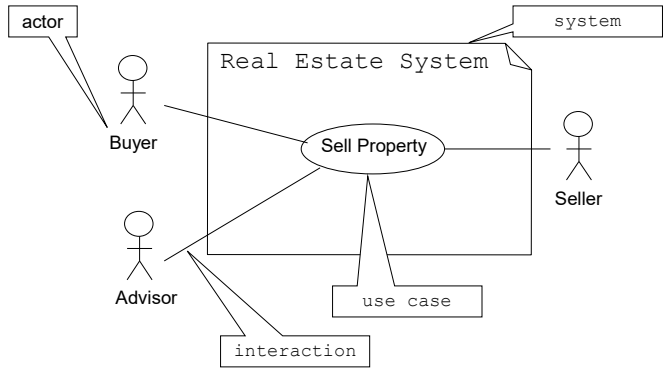
Use Case Diagram

- Use case diagrams are created to visualize the relationships between actors and use cases
- Show the interaction between use cases, which represent system functionality, and actor, which represent the people or system that provide or receive information from the system.



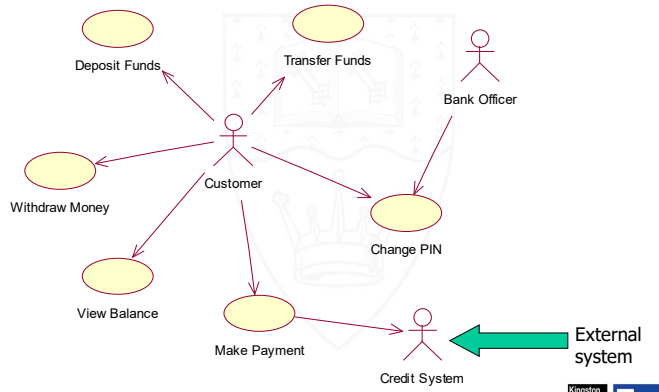
76

Use Case Diagram



77

ATM Use Cases



78

Associations in Use Case Diagram

- *Associations* can exist
  - between an actor and a use case,
  - between use cases
  - between actors
- *Types of Use Case Associations*
  - *Communicates* between actor and use case
    - named or unnamed relationship showing participation of actor in use case, use a solid line connecting actor to use case
  - *Generalization* between actors

79

Associations in Use Case Diagram

- adornments = Stereotyped Associations between use cases
  - **<<extend>>**

    indicates relationship between use cases in which a special use case (the non-arrow end) extends an original use case (the arrow end)
  - **<<include>>**

    reuses steps in a use case instead of cut-and-pasting steps into multiple use case documents, by pulling out common steps into a new use case and specifying with an arrowed line the <<include>> association between this new use case and those use cases requiring the steps
  - **<<uses>>**

    An instance of the source use case includes behavior described by the target, Shows a stereotyped **generalization** relationship between use cases

80

Relationship	Function	Notation
association	The communication path between an actor and a use case that it participates in	———
extend	The insertion of additional behavior into a base use case that does not know about it	«extend» - - - ->
use case generalization	A relationship between a general use case and a more specific use case that inherits and adds features to it	——▷
include	The insertion of additional behavior into a base use case that explicitly describes the insertion	«include» - - - ->

81

Use Case Relationships

- **Include,** when you have a chunk of behaviors that is similar across more than one use case
- **Generalization,** when you have one use case that is similar to another use case but does a bit more
- **Extend,** similar to generalization but with more rules to it.

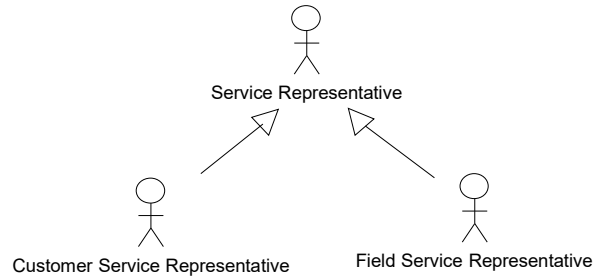
82

Rules

- Use *include* when you are repeating yourself in two or more separate use cases and you want to avoid repetition.
- Use *generalization* when you are describing a variation on normal behavior and you wish to describe it casually.
- Use *extend* when you are describing a variation on normal behavior and you wish to use the more controlled form, declaring your extension points in your base use case.

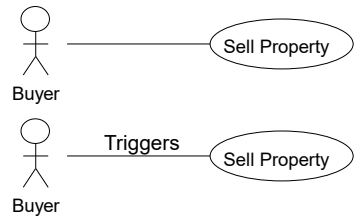
83

Example of Generalization between Use Case Actors



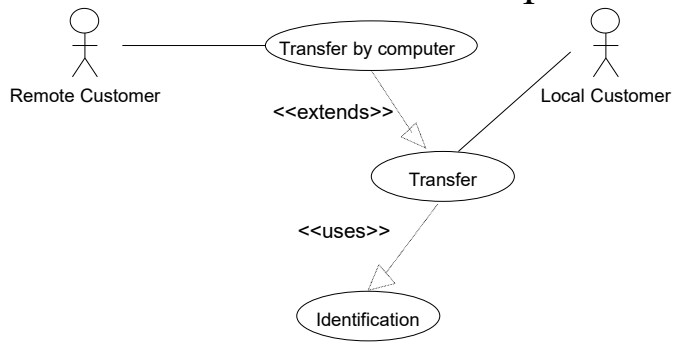
84

Example of Communicates Use Case Relationship



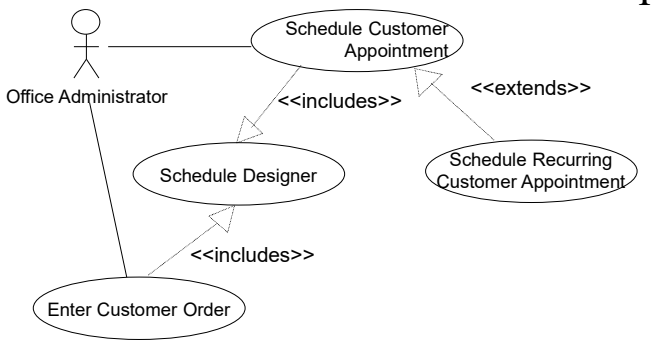
85

Example <<uses>> and <<extends>> Use Case Relationships



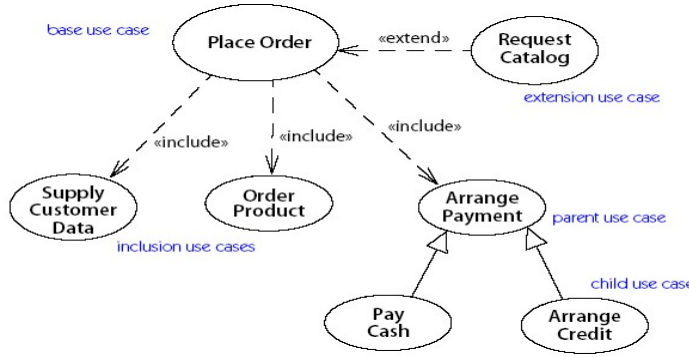
86

Example <<include>> and <<extends>> Use Case Relationships



87

More example



88

Library Case Exercise

Problem Statement:
A library contains video cassettes which can be borrowed by registered uses. User can borrow at most four items. User can reserve a cassette which item is on loan. User can reserve at most 2 items. User can borrow a returned item again if it is not reserved. User can search for a particular item in a library by a keyboard.

Exercise: Create a Use Case Diagrams and Scenarios

89

Class Diagram

- A class diagram shows the existence of classes and their relationships in the logical view of a system
- UML modeling elements in class diagrams
  - Classes and their structure and behavior
  - Association, aggregation, dependency, and inheritance relationships
  - Role names

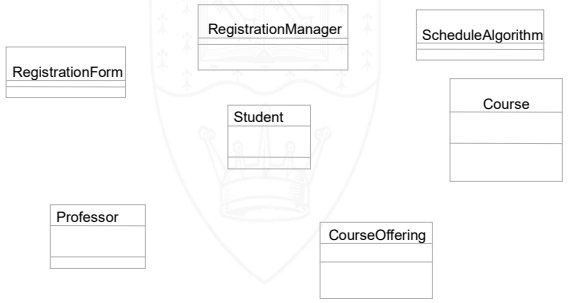
90

Classes

- A class is a collection of objects with common structure, common behavior, common relationships and common semantics
- Classes are found by examining the objects in sequence and collaboration diagram
- A class is drawn as a rectangle with three compartments
- Classes should be named using the vocabulary of the domain
  - Naming standards should be created
  - e.g., all classes are singular nouns starting with a capital letter

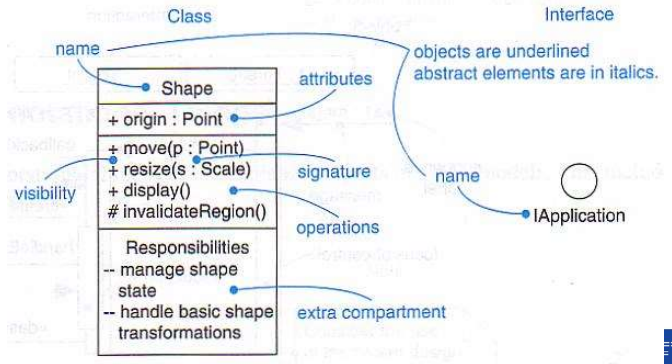
91

Example



92

More example



93

Relationship

- Relationships provide a pathway for communication between objects
- Sequence and/or collaboration diagrams are examined to determine what links between objects need to exist to accomplish the behavior -- if two objects need to "talk" there must be a link between them
- Three types of relationships are:
  - Association
  - Aggregation
  - Dependency

94

Relationship

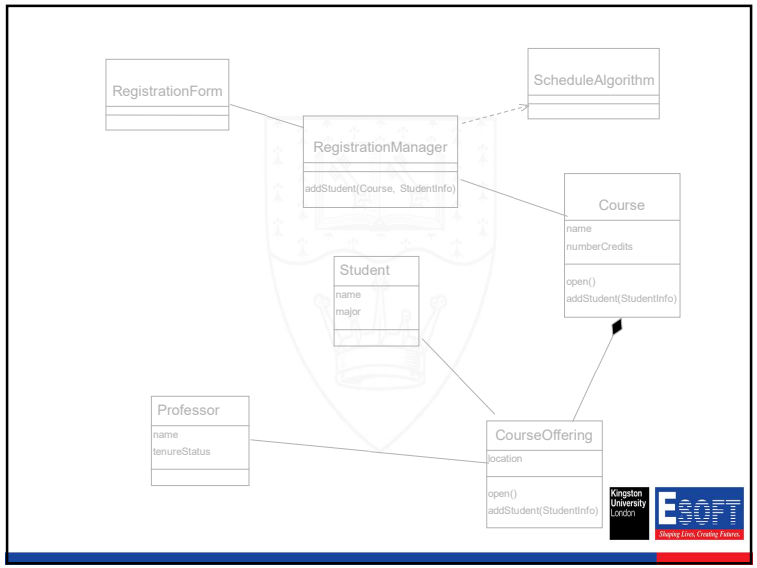
- An association is a bi-directional connection between classes
  - An association is shown as a line connecting the related classes
- An aggregation is a stronger form of relationship where the relationship is between a whole and its parts
  - An aggregation is shown as a line connecting the related classes with a diamond next to the class representing the whole
- A dependency relationship is a weaker form of relationship showing a relationship between a client and a supplier where the client does not have semantic knowledge of the supplier
- A dependency is shown as a dashed line pointing from the client to the supplier

95

Finding relationship

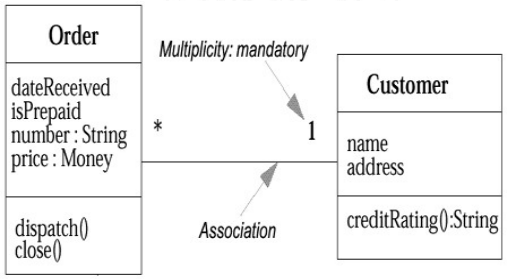
- Relationships are discovered by examining interaction diagrams
  - If two objects must "talk" there must be a pathway for communication

96



97

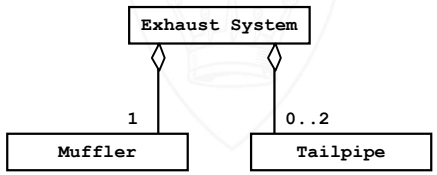
Associations



98

Aggregation

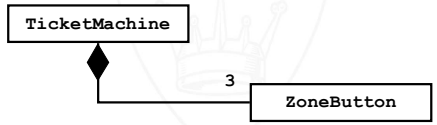
- Associations denote relationships between classes
- ***Aggregation*** "consists of" hierarchy.
- The ***aggregate*** is the parent class, the ***components*** are the children class.



99

Composition

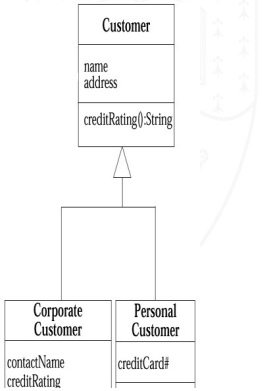
- A solid diamond denote ***composition,*** a strong form of aggregation where components cannot exist without the aggregate.



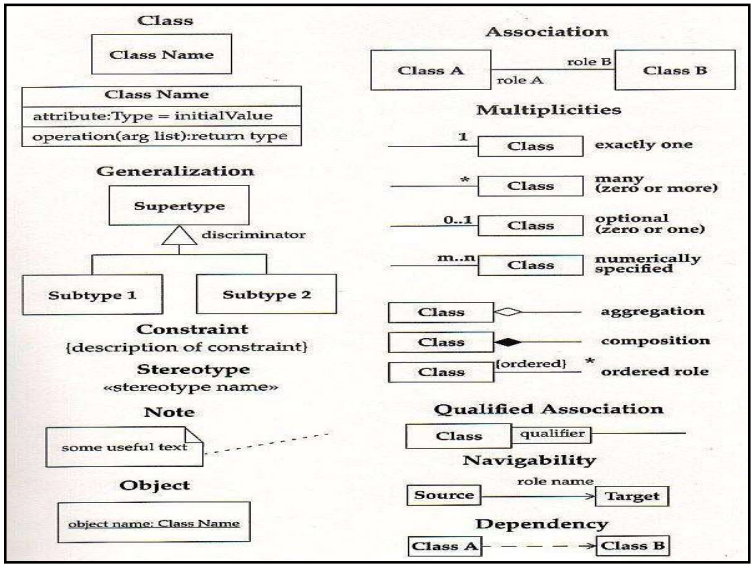
100

Generalization

Is a kind of

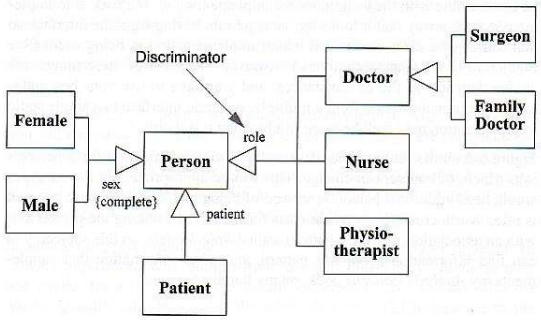


101



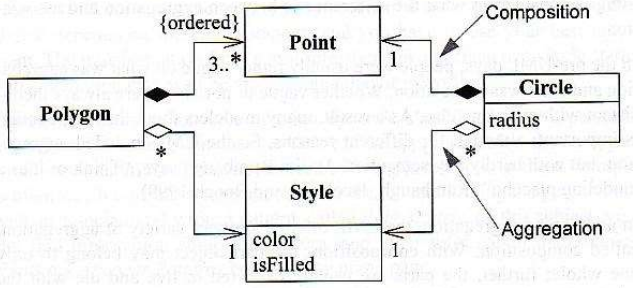
102

More examples



103

More examples



104

Sequence diagram

- A sequence diagram displays object interactions arranged in a time sequence
- It is used to show the flow of functionality through a use case

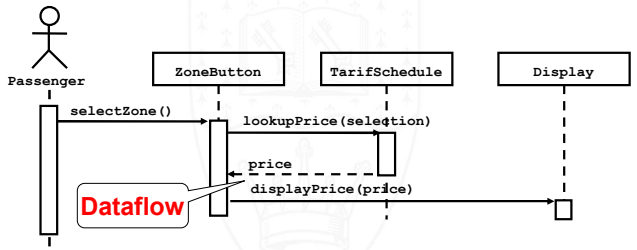
105

Sequence Diagrams

- Used during requirements analysis
  - To refine use case descriptions
  - to find additional objects ("participating objects")
- Used during system design
  - to refine subsystem interfaces
- ***Classes*** are represented by columns
- ***Messages*** are represented by arrows
- ***Activations*** are represented by narrow rectangles
- ***Lifelines*** are represented by dashed lines

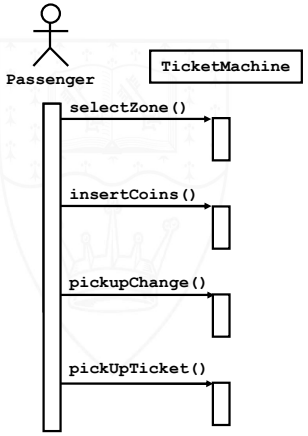
106

Nested Messages



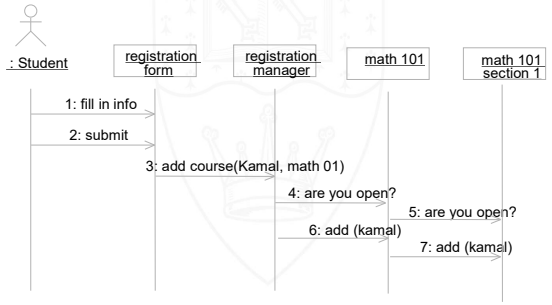
- The source of an arrow indicates the activation which sent the message
- An activation is as long as all nested activations

107



108

More example



109

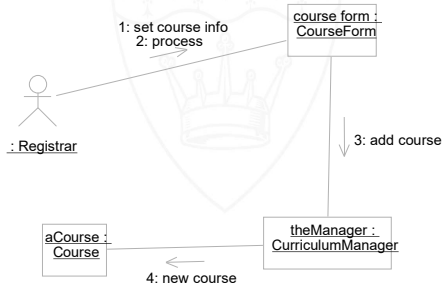
Collaboration Diagrams

- Show exactly the same functions as the sequence diagram
- But it shows in different way and different purpose

110

Collaboration diagram

- A collaboration diagram displays object interactions organized around objects and their links to one another



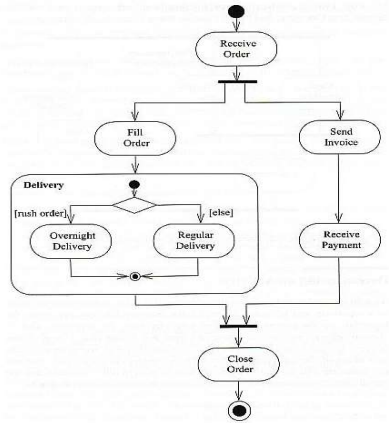
111

Activity Diagrams

- An activity diagram shows flow control within a system

112

Example



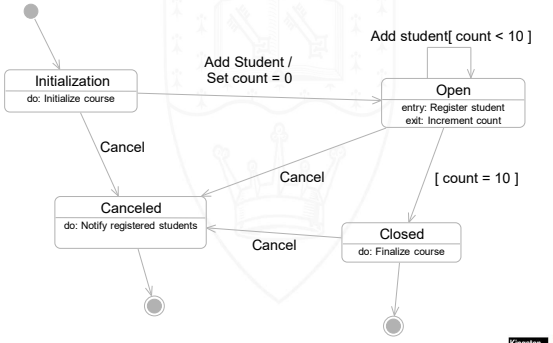
113

State diagram

- A state transition diagram shows
  - The life history of a given class
  - The events that cause a transition from one state to another
  - The actions that result from a state change
- State transition diagrams are created for objects with significant dynamic behavior

114

Example



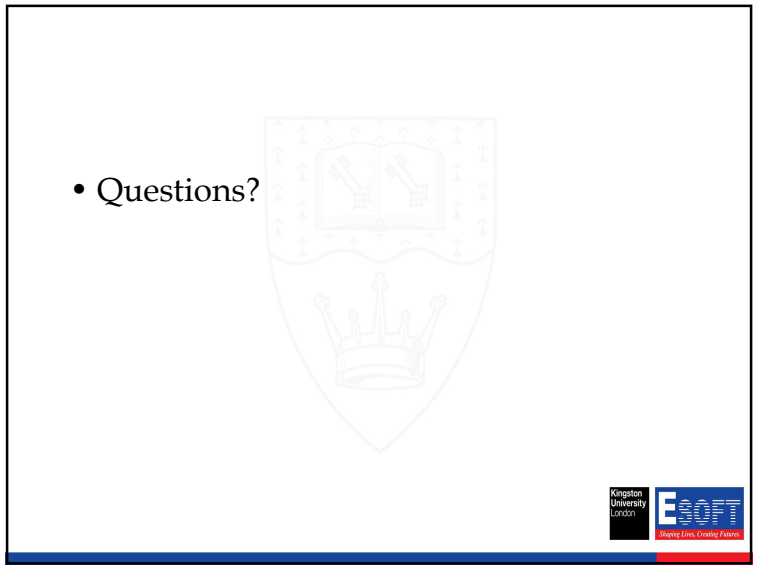
115

Object Modeling with the Unified Modeling Language

- CASE Tools
  - Object modeling requires many types of diagrams to represent the proposed system
  - Creating the diagrams by hand is time-consuming and tedious, so systems analysts rely on CASE tools to speed up the process and provide an overall framework for documenting the system components

116

- Questions?



117