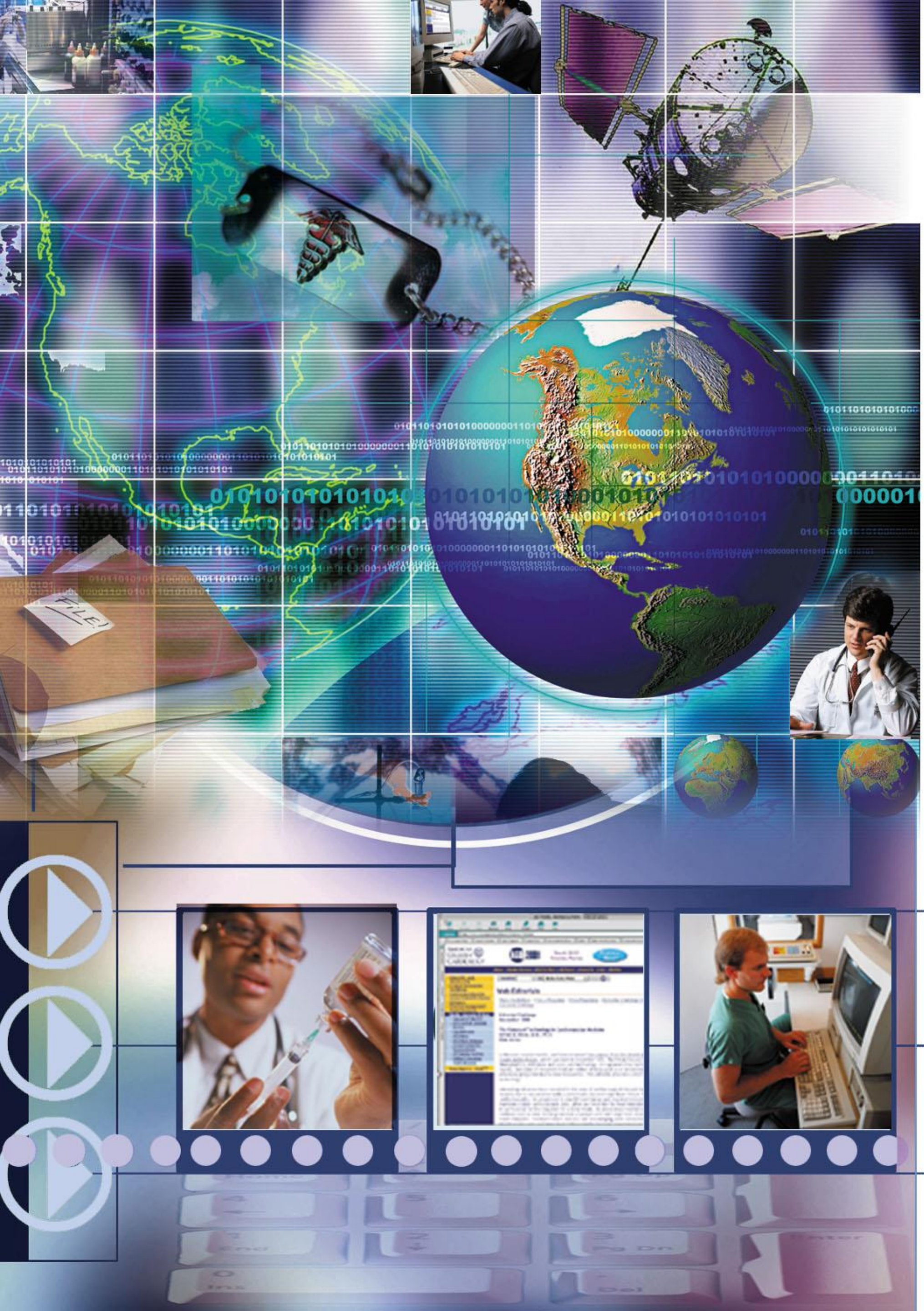


Chapter 4: DESIGN WITH OBJECTS

By:

Ts. Azaliza binti Zainal

OBJECT-ORIENTED DESIGN – THE BRIDGE BETWEEN ANALYSIS AND PROGRAMMING

- Bridge between users' requirements and new system's programming.
- Object-oriented design is process by which detailed object-oriented models are built.
- Programmers use design to write code and test new system.
- User interface, network, controls, security, and database require design tasks and models.

OBJECT-ORIENTED CONCEPTS

- The term **object-oriented** was first introduced in connection to object-oriented programming and Smalltalk, although the object-oriented concepts of information hiding and inheritance have earlier origins.
- Object-oriented concepts are considered important in software development because they address fundamental issues of adaptation and evolution.

Objects and Classes

- Set of objects that cooperate to accomplish result
- Object contains program logic and necessary attributes in a single unit
- Objects send each other messages and collaborate to support functions of main program
- OO systems designer provides detail for programmers
  - Design class diagrams, interaction diagrams, and some state machine diagrams

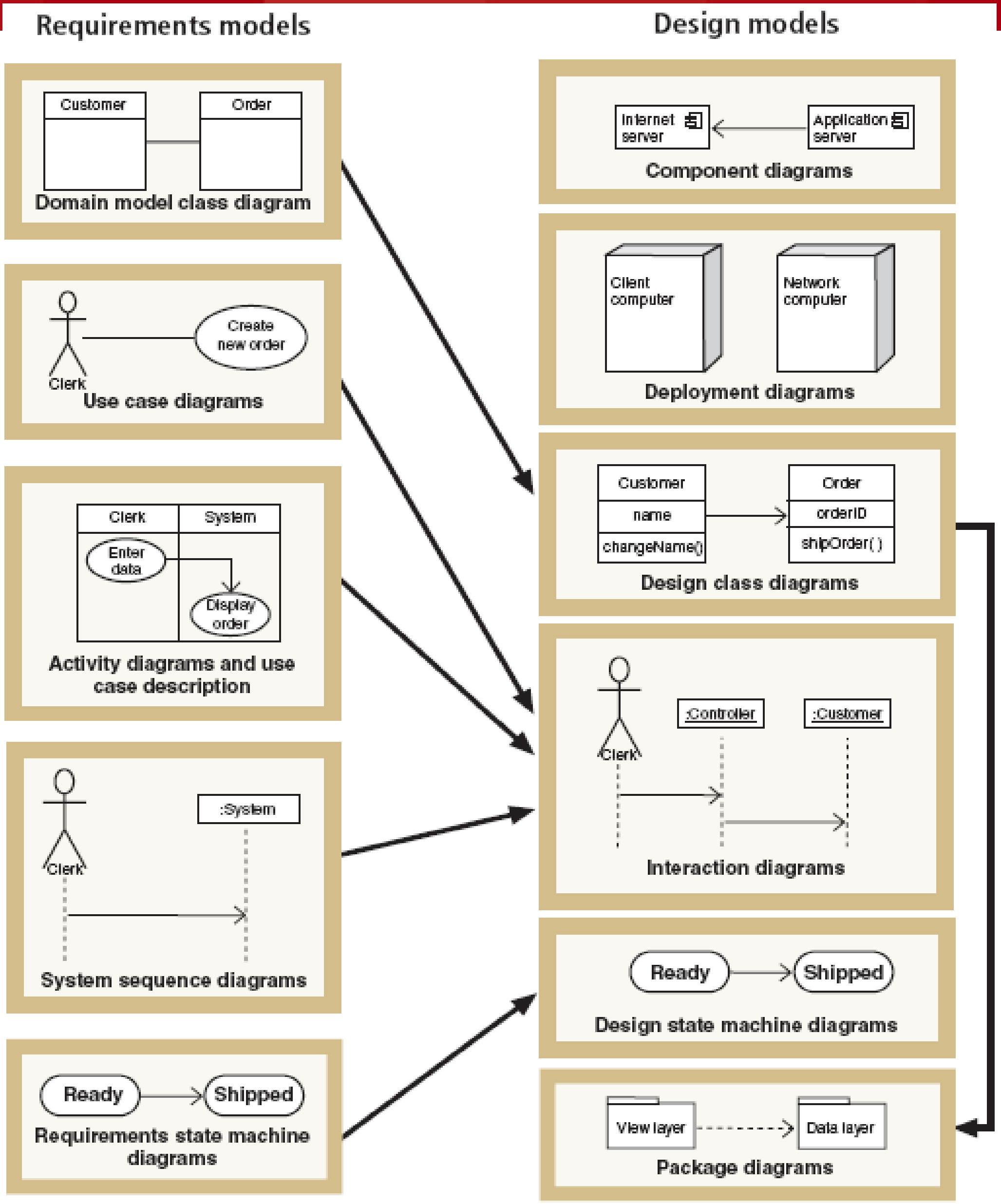
How is Design Different from Analysis?

- The analyst seeks to understand the organization, its requirements and its objectives.
- The designer seeks to specify a system that will fit the organization, provide its requirements effectively and assist it to meet its objectives.

Object-Oriented Design Processes and Models

- **Diagrams developed for analysis/requirements**
  - Use case diagrams, use case descriptions and activity diagrams, domain model class diagrams, and system sequence diagrams
- **Diagrams developed for design**
  - Component diagrams and Deployment diagrams
  - Interaction diagrams and package diagrams
  - Design class diagrams

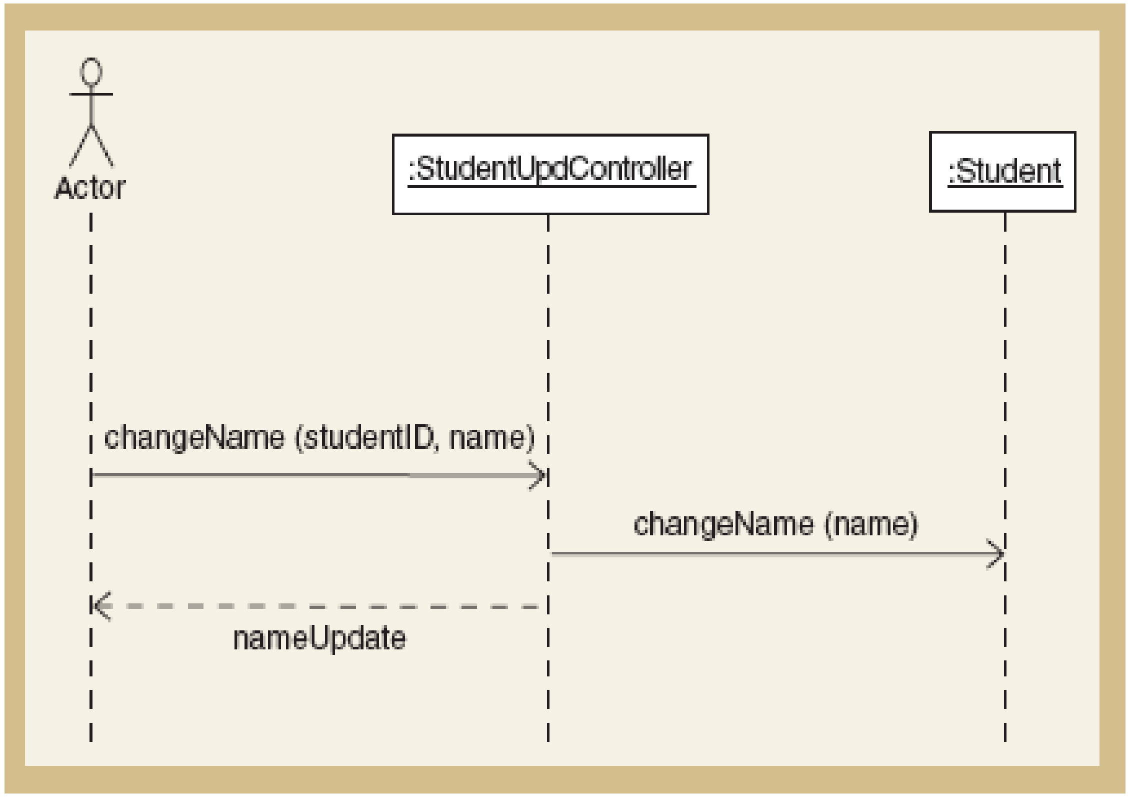
Design Models with Their Respective Input Models



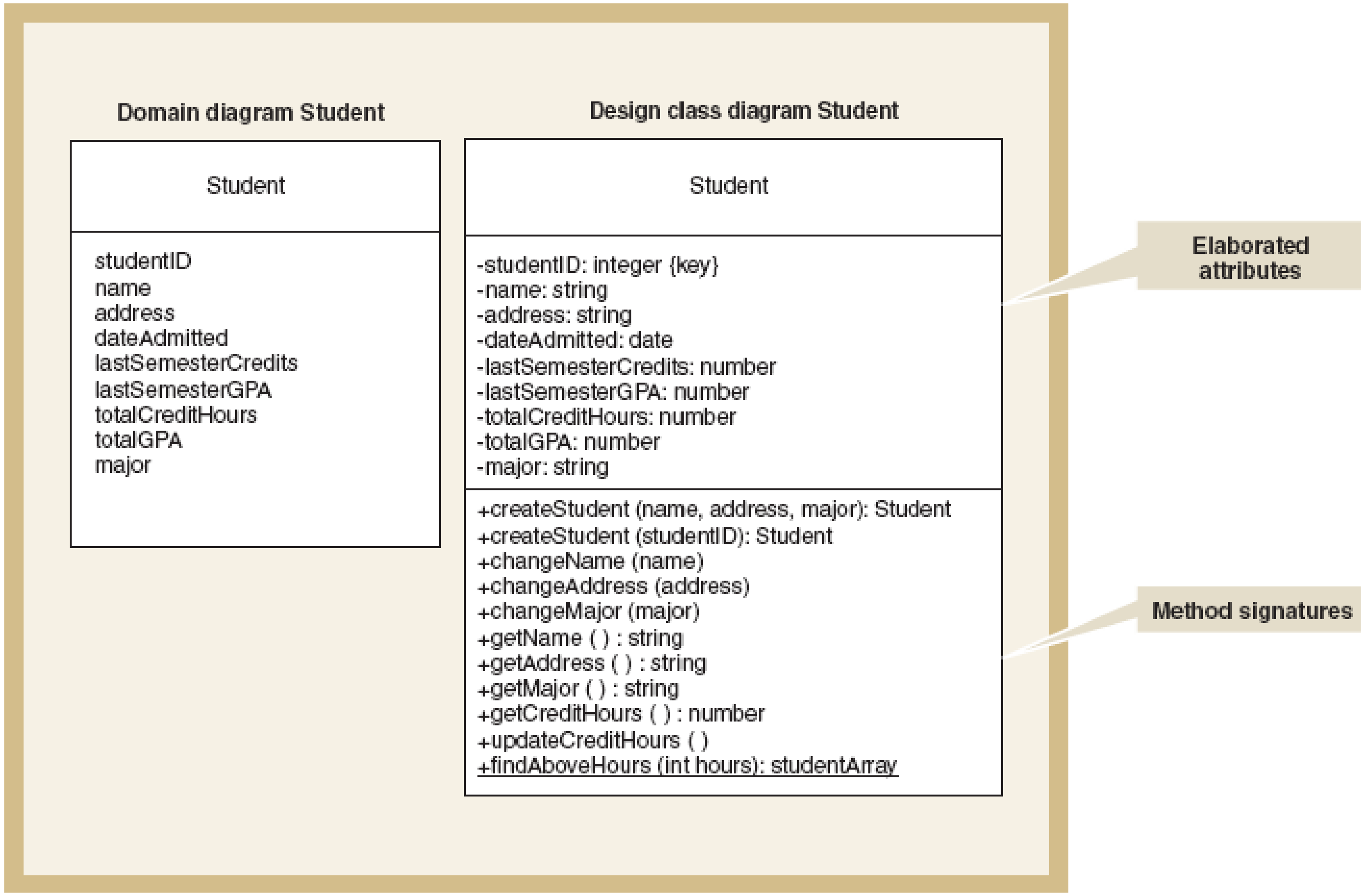
FUNDAMENTAL PRINCIPLES OF OBJECT-ORIENTED DETAILED DESIGN

- **Design class diagrams and detailed sequence diagrams**
  - –Use each other as inputs and are developed in parallel
- **Sequence diagrams define the interactions between objects in order to execute a use case.**
  - –Interactions are called messages
  - –Correspond to method calls in programming language
- **Design Classes show attributes and method signatures**

SAMPLE SEQUENCE DIAGRAM



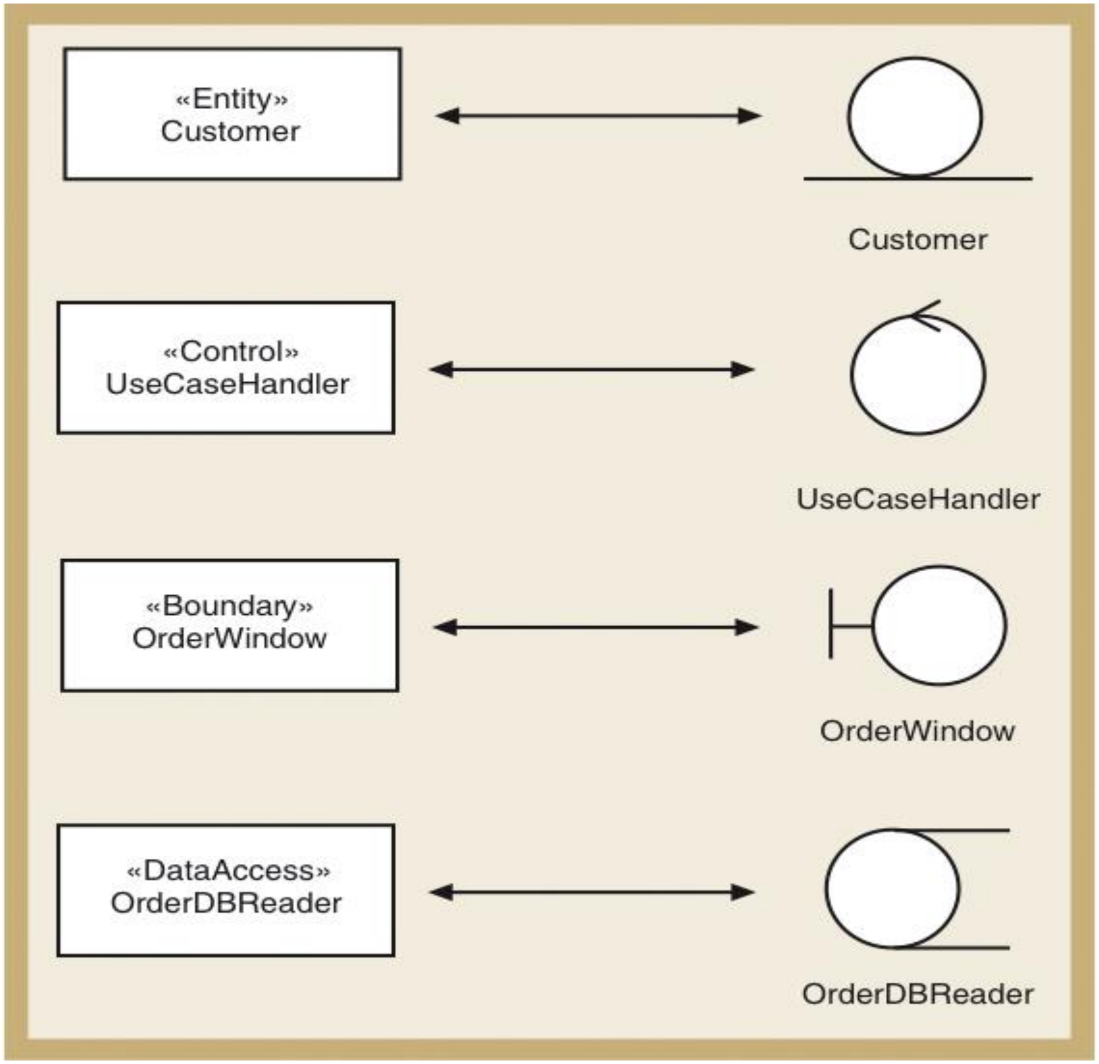
SAMPLE DESIGN CLASS



DESIGN CLASS SYMBOLS

- UML does not distinguish between design class notation and domain model notation
- Domain model class diagram shows conceptual classes in users' work environment
- Design class diagram specifically defines software classes
- UML uses stereotype notation to categorize a model element by its characteristics

STANDARD STEREOTYPES FOUND IN DESIGN



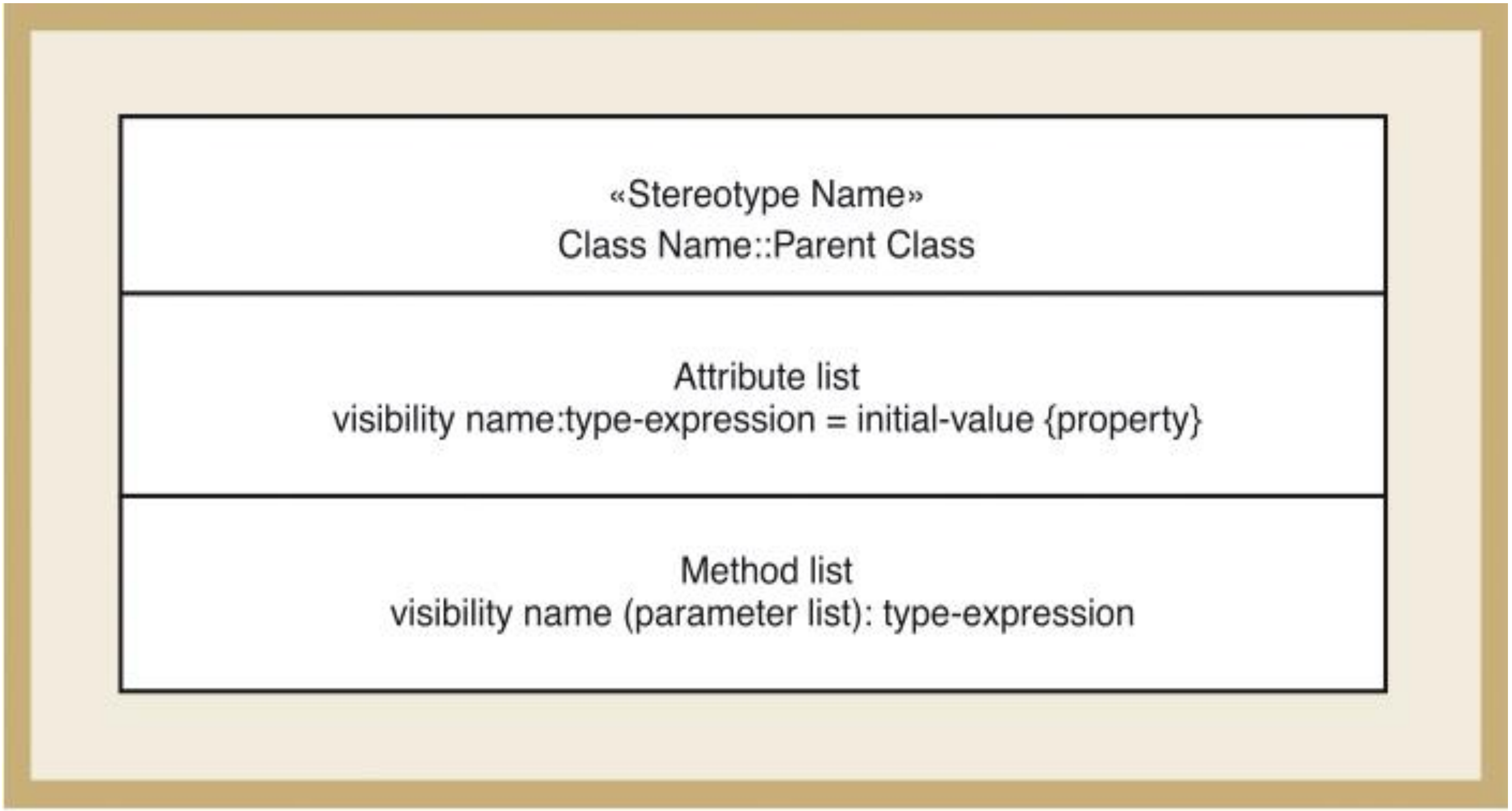
STANDARD DESIGN CLASSES

- Entity – design identifier for problem domain class
  - Persistent class – exists after system is shut down
- Control – mediates between boundary and entity classes, between the view layer and domain layer
- Boundary – designed to live on system's automation boundary, touched by users
  - User interface and windows classes
- Data access – retrieves data from and sends data to database

DESIGN CLASS NOTATION

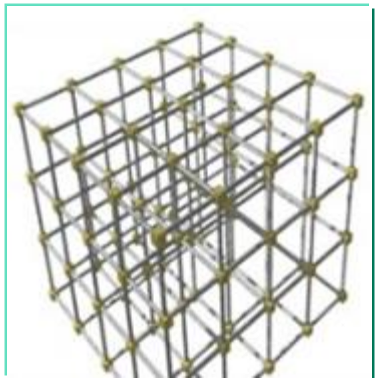
- Name – class name and stereotype information
- Attribute visibility (private or public) – attribute name, type-expression, initial-value, property
- Method signature – information needed to invoke (or call) the method
  - Method visibility, method name, type-expression (return parameter), method parameter list (incoming arguments)
  - Overloaded method – method with same name but two or more different parameter lists
  - Class-level method – method associated with class instead of each object (static or shared method), denoted by an underline

NOTATION USED TO DEFINE A DESIGN CLASS


«Stereotype Name»
Class Name::Parent Class
Attribute list
visibility name:type-expression = initial-value {property}
Method list
visibility name (parameter list): type-expression

WHAT IS FRAMEWORK?

- "A framework is a skeleton of code used for developing an application with ease".
- A framework is a suite of package using which application will implement its function and non-functional requirements.
- Every framework has its own purpose, best-practices, restrictions and limitations.
- Framework in general specifies collection of classes and interfaces that are designed to work together to handle a particular type of problem.



FRAMEWORK

- An application framework is a specific set of classes that cooperate closely with each other and together embody a reusable design for a category of problems.

  - e.g.: Java APIs (Applet, Thread, etc)
  - e.g.: MFC, JFC, etc.

- A framework dictates the architecture of an application and can be customized to get an application. (e.g.: Java Applets)

EXAMPLES OF FRAMEWORK

- Struts and JSF - Java Server Faces are frameworks for building Web Interfaces.
- Toplink and Hybernate are frameworks for object persistence.
- Log4J is a framework for auditory. You can use Log4J for implementing technical logs and business audit logs.
- J2EE is framework for building Enterprise Applications.
- LEAF - Lucasian Enterprise Application Framework is a white-box framework for building Enterprise Java Applications with high productivity in development and robustness in production environments.

OBJECT-ORIENTED FRAMEWORKS

- **In the object oriented paradigm, a framework is composed of a library of classes.**
  - The API is defined by the set of all public methods of these classes.
  - Some of the classes will normally be abstract and there are often many Interfaces

- **Example:**
  - A framework for payroll management
  - A framework for frequent buyer clubs
  - A framework for university registration
  - A framework for e-commerce web sites

OBJECT-ORIENTED DESIGN PRINCIPLES

Some Fundamental Design Principles

- Encapsulation – each object is self-contained unit that includes data and methods that access data
- Object reuse – designers often reuse same classes for windows components
- Information hiding – data associated with object is not visible to outside world

Some Fundamental Design Principles (continued)

- Coupling – qualitative measure of how closely classes in a design class diagram are linked
  - Number of navigation arrows in design class diagram or messages in a sequence diagram
  - Loosely coupled – system is easier to understand and maintain
- Cohesion – qualitative measure of consistency of functions within a single class
  - Separation of responsibility – divide low cohesive class into several highly cohesive classes
  - Highly cohesive – system is easier to understand and maintain and reuse is more likely

Some Fundamental Design Principles (continued)

- Protection from variations – parts of a system that are unlikely to change are segregated from those that will
- Indirection – an intermediate class is placed between two classes to decouple them but still link them
- Object responsibility – Objects are responsible for system processing
  - Responsibilities include knowing and doing

OBJECT-ORIENTED DEVELOPMENT

- **Object-oriented analysis, design and programming, are related but distinct.**
- **OOA is concerned with developing an object model of the application domain.**
- **OOD is concerned with developing an object-oriented system model to implement requirements.**
- **OOP is concerned with realising an OOD using an OO programming language such as Java or C++.**

UNIFIED MODELING LANGUAGE (UML)

- Several different notations for describing object oriented designs were proposed in the 1980s and 1990s.
- The Unified Modeling Language is an integration of these notations.
- It describes notations for a number of different models that may be produced during OO analysis and design.
- It is now a *de facto* standard for OO modelling.

CLASS DIAGRAM

- UML provides class diagrams for illustrating classes, interfaces and their associations.
- Class diagrams are used for static object modelling.
- Note that class diagrams are also used for visualizing domain models.

CLASS DIAGRAM

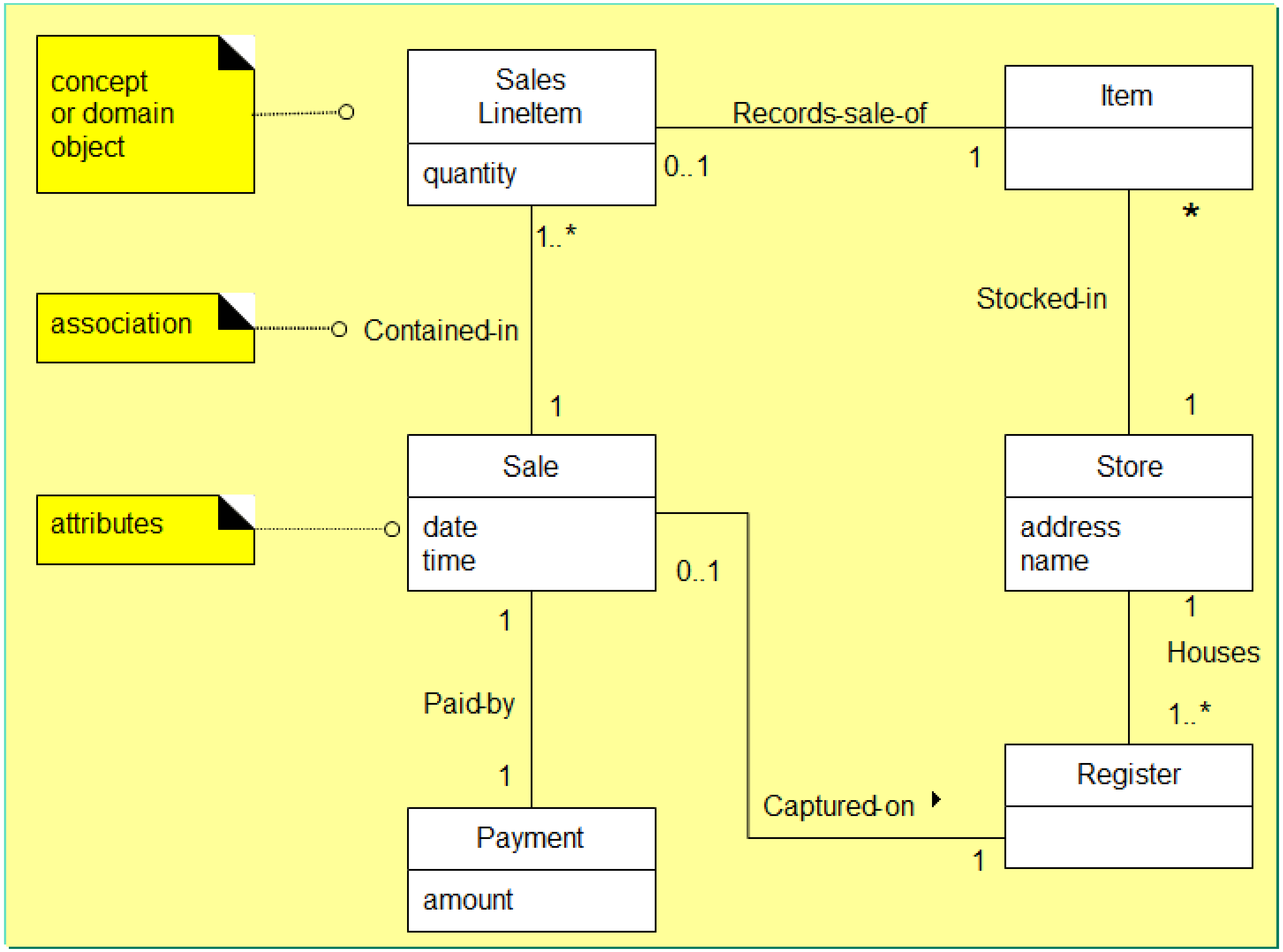
- Class diagrams are the backbone of almost every object oriented method, including UML.
- They describe the classes of the system, their inter-relationships, and the operations and attributes of the classes.
- Classes represent an abstraction of entities with common characteristics. Associations represent the relationships between classes.
- Illustrate classes with rectangles divided into compartments. Place the name of the class in the first partition (centered, bolded, and capitalized), list the attributes in the second partition, and write operations into the third.

CLASS DIAGRAM

- A domain model is a visual representation of conceptual classes or real-world objects in a domain of interest.
- In UML, a **domain model** is illustrated **with a set of class diagrams** in which no operations are defined.
- It shows:

- conceptual classes (concepts / domain objects)
- associations between conceptual classes
- attributes of conceptual classes

CLASS DIAGRAM: EXAMPLE



CLASS DIAGRAM: USEFUL GUIDELINES

- Focus on "need-to-know" associations.
- It is more important to identify conceptual classes than to identify associations.
- Too many associations tend to confuse a domain model.
- Avoid showing redundant or derivable associations.

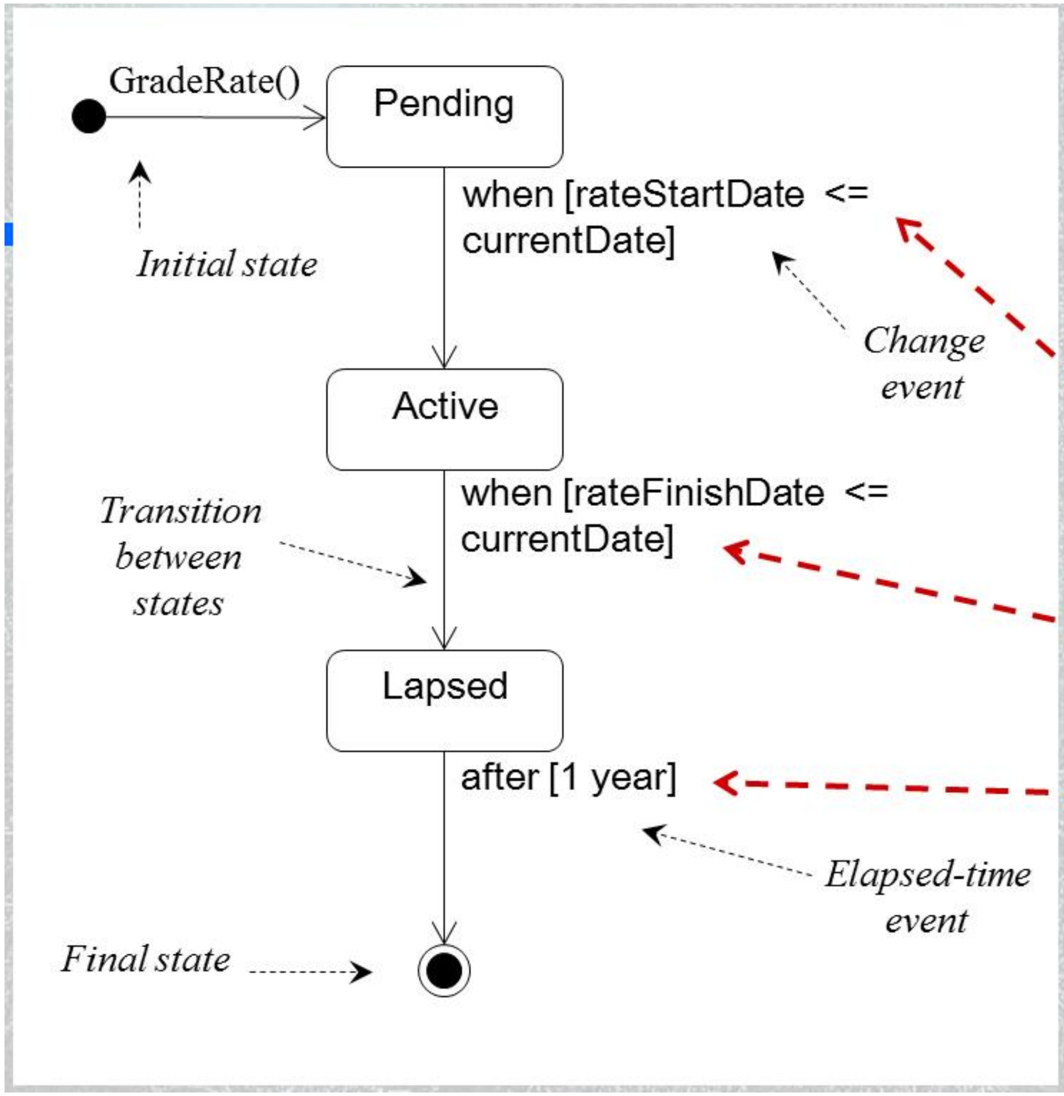
STATECHART DIAGRAM

- ‘A state is a condition during the life of an object or an interaction during which it satisfies some condition, performs some action or waits for some event. Conceptually, an object remains in a state for an interval of time. However, the semantics allow for modelling "flow-through" states which are instantaneous as well as transitions that are not instantaneous.’
- A statechart diagram shows the behavior of classes in response to external stimuli. This diagram models the dynamic flow of control from state to state within a system.

STATECHART DIAGRAM: EXAMPLE

- The current state of an object is determined by the current value of the object's attributes and the links that it has with other objects
- For example the class *StaffMember* has an attribute *startDate* which determines whether a *StaffMember* object is in the probationary state
- The current state of a *GradeRate* object can be determined by the two attributes *rateStartDate* and *rateFinishDate*
- An enumerated state variable may be used to hold the object state, possible values would be Pending, Active or Lapsed

STATECHART DIAGRAM



Statechart

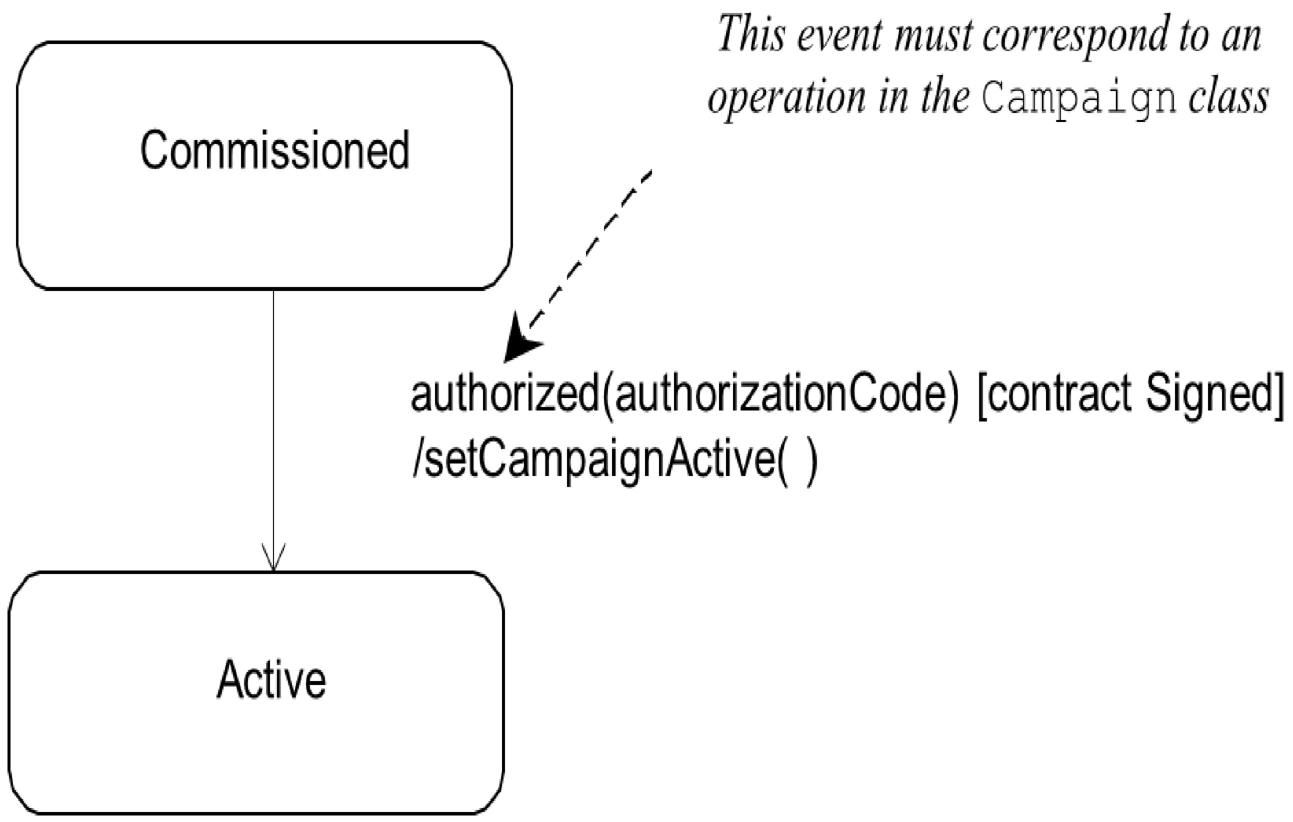
Statechart for the class GradeRate.

Movement from one state to another is dependent upon events that occur with the passage of time.

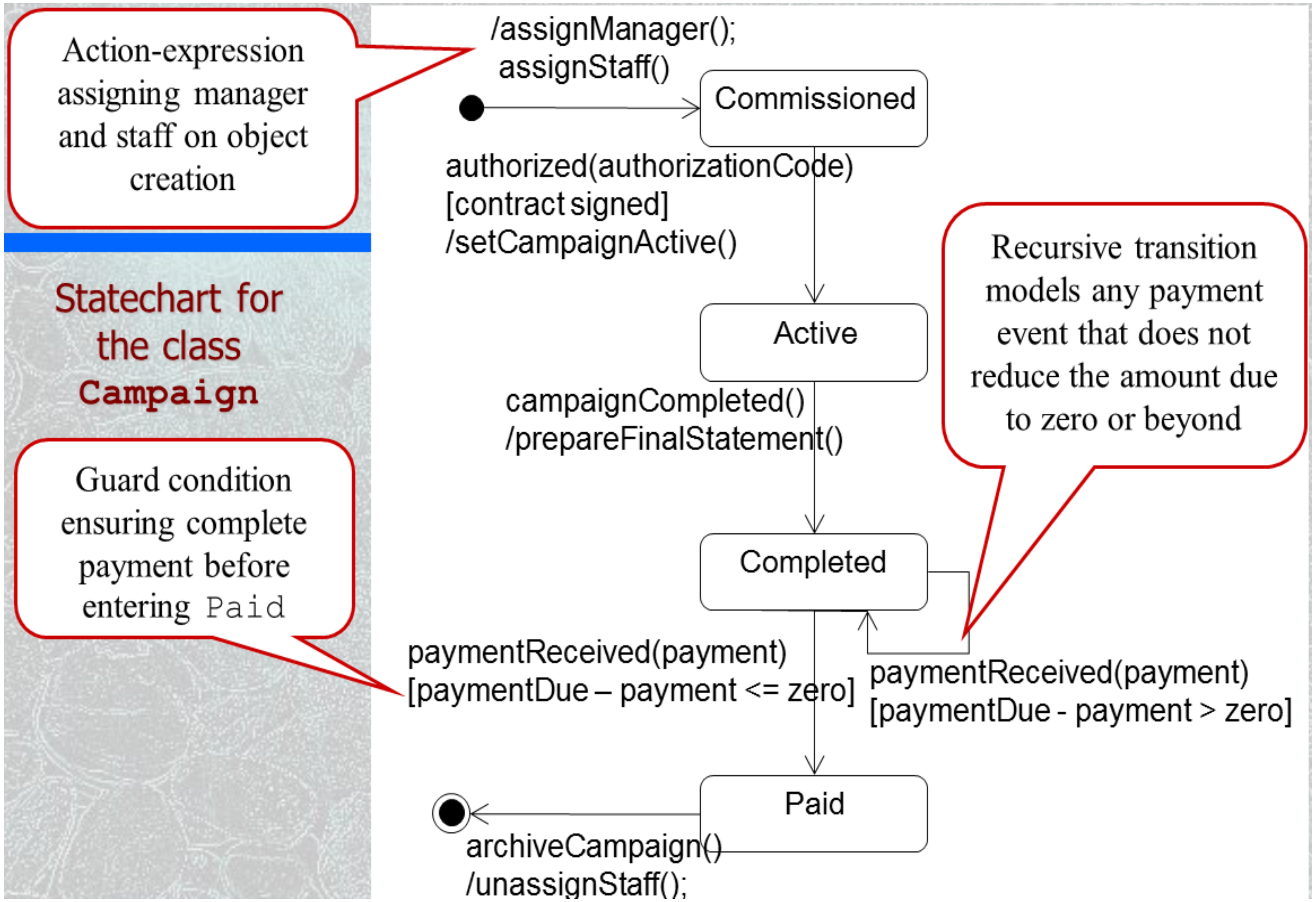
STATECHART DIAGRAM

- Types of Event
  - A change event occurs when a condition becomes true
  - A call event occurs when an object receives a call to one of its operations either from another object or from itself
  - A signal event occurs when an object receives a signal (an asynchronous communication)
  - An elapsed-time event is caused by the passage of a designated period of time after a specified event (frequently the entry to the current state)

STATECHART DIAGRAM


This event must correspond to an operation in the Campaign class
Commissioned
authorized(authorizationCode) [contract Signed] /setCampaignActive( )
Active

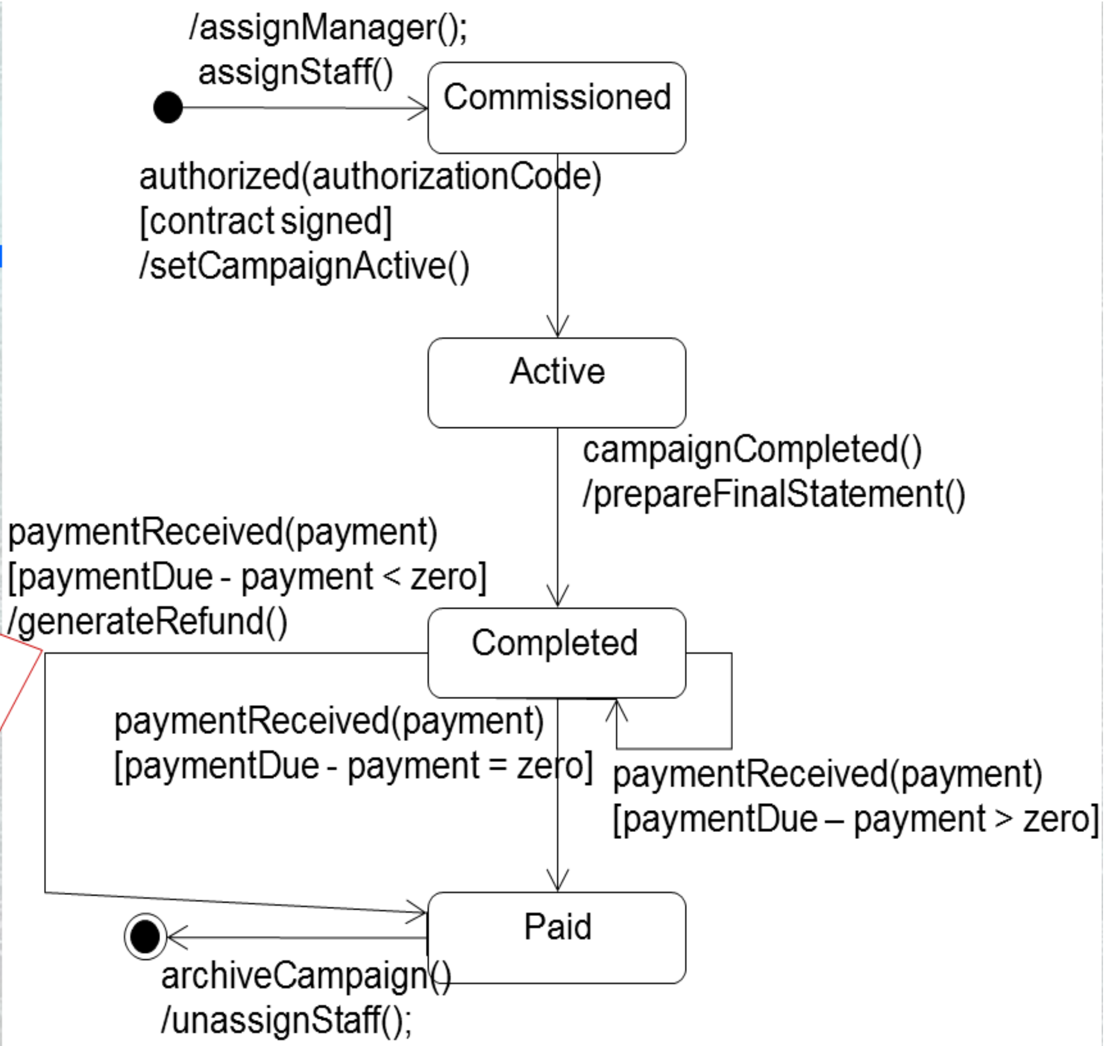
STATECHART DIAGRAM


Action-expression assigning manager and staff on object creation
Statechart for the class Campaign
Guard condition ensuring complete payment before entering Paid
/assignManager(); assignStaff()
Commissioned
authorized(authorizationCode) [contract signed] /setCampaignActive()
Active
campaignCompleted() /prepareFinalStatement()
Completed
Recursive transition models any payment event that does not reduce the amount due to zero or beyond
paymentReceived(payment) [paymentDue – payment <= zero]
paymentReceived(payment) [paymentDue - payment > zero]
Paid
archiveCampaign() /unassignStaff();

STATECHART DIAGRAM

A revised statechart for the class **Campaign**

If the user requirements were to change, so that an overpayment is now to result in the automatic generation of a refund, a new transition is added



Q & A

THANK YOU FOR YOUR ATTENTION

By:
Ts. Azaliza Zainal
Software Engineering Cluster, Computing Department, FCVAC