

Object-Oriented Analysis and Design with UML

Overview

Object-Oriented Concepts

UML Basics

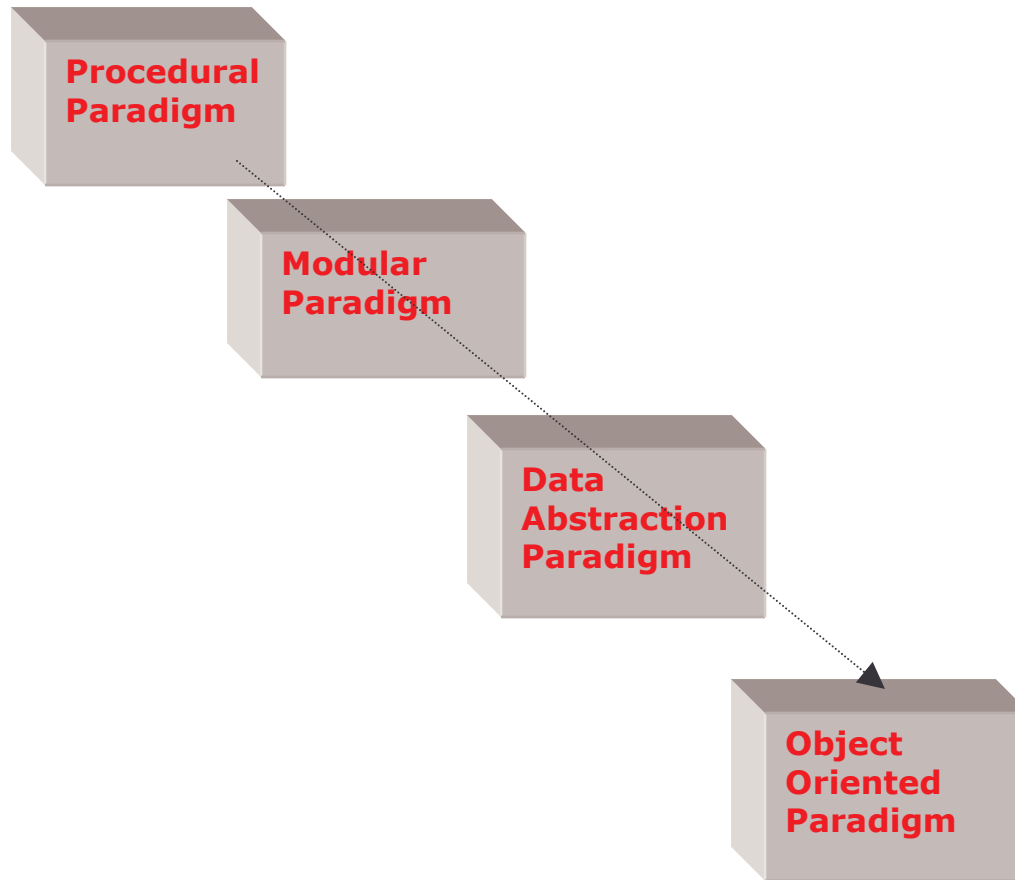
Object Oriented Concepts

Overview: OO Concepts

- 1) Background and Principles of Object Orientation
- 2) Object Oriented Concepts
- 3) Object Oriented Analysis (OOA)
- 4) Object Oriented Design (OOD)

Background and Principles

The Road to OO



What is OO?

Definition

Object orientation is about viewing and modelling the world (or any system) as a set of interacting and interrelated objects.

An approach characterized by the following features:

- 1) views the universe of discourse as consisting of interacting objects
- 2) describes and builds systems consisting of representation of objects

Principles of OO

- 1) Abstraction
- 2) Encapsulation
- 3) Modularity
- 4) Hierarchy

Abstraction

A model that includes most important aspects of a given system while ignoring less important details.

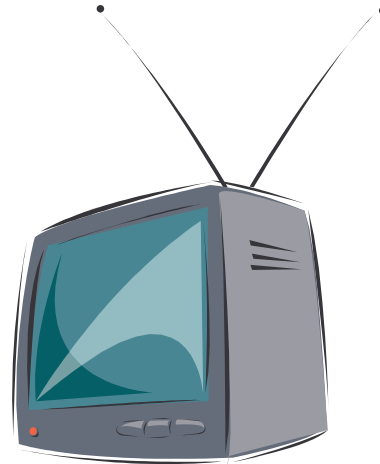
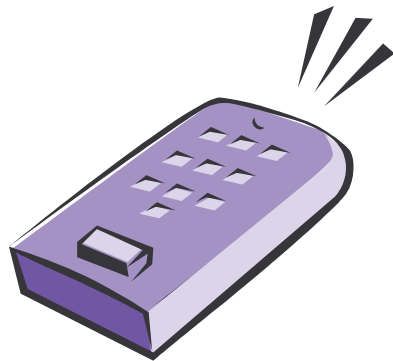
Abstraction allows us to manage complexity by concentrating on essential characteristics that makes an entity different from others.



An example of an order processing abstraction

Encapsulation 1

- 1) Encapsulation separates implementation from users or clients.
- 2) Clients depend on interface.



Courtesy Rational Software

Modularity

Modularity deals with the process of breaking up complex systems into small, self contained pieces that can be managed easily.

Order
Processing
System



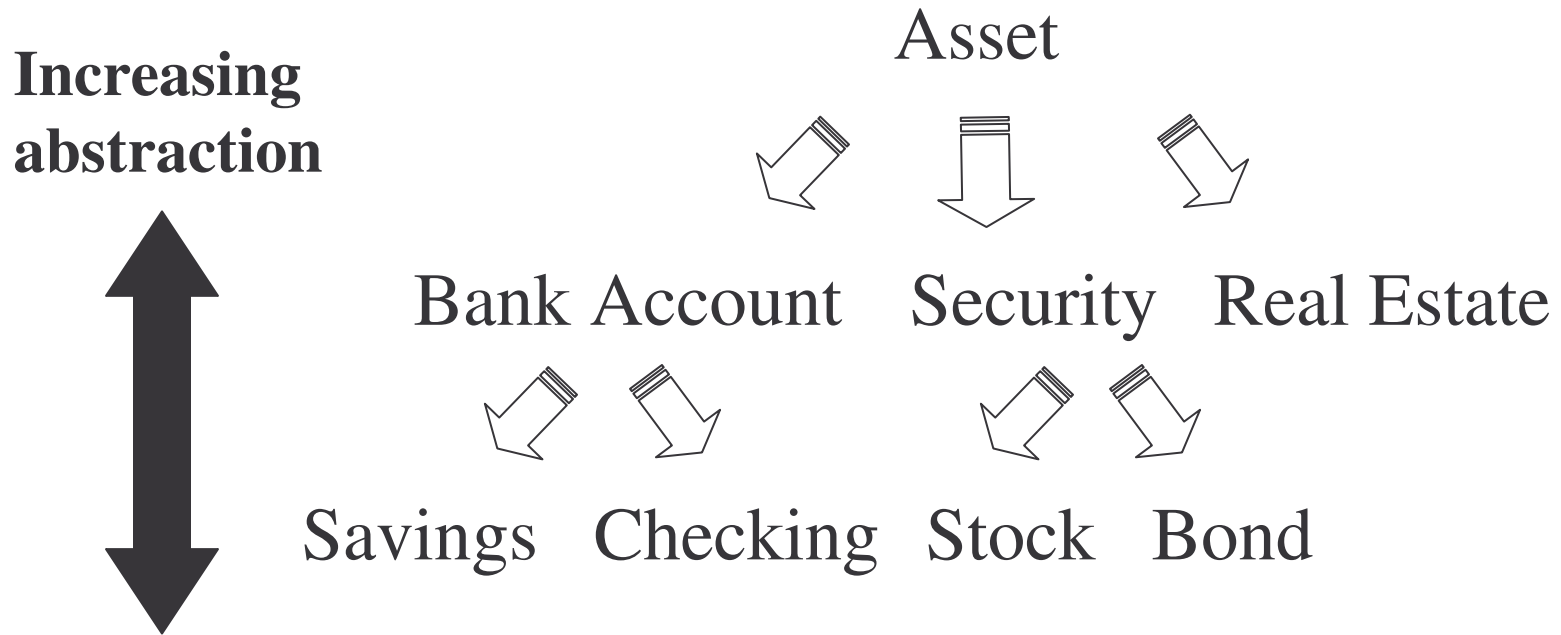
Order Entry

Order Fulfillment

Billing

Hierarchy

Is an ordering of abstractions into a tree like structure.



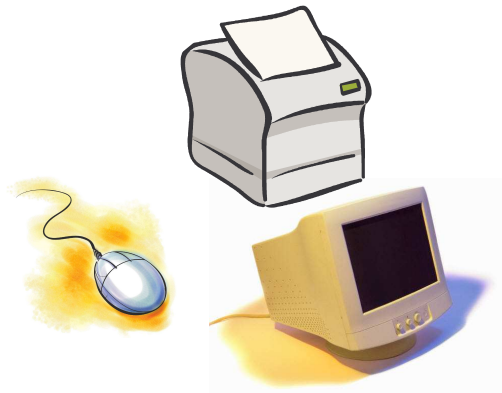
Concepts

Objects

What is an **object**?

- 1) any abstraction that models a single thing
- 2) a representation of a specific entity in the real world
- 3) may be tangible (physical entity) or intangible
- 4) examples: specific citizen, agency, job, location, order ...

Example: Objects



Object Definition

Two aspects: information and behaviour

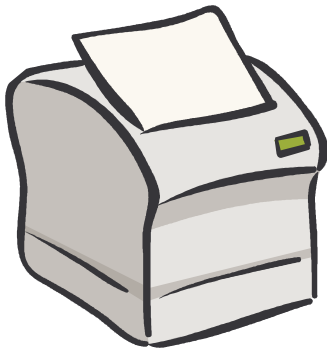
Information:

- 1) has a unique identity
- 2) has a description of its structure or the information that is used to create it
- 3) has a state representing its current condition, e.g. values of some of its features

Behaviour:

- 1) what can the object do?
- 2) what can be done to it?

Example: Object Definition



1) **information:**

- a) serial number
- b) model
- c) speed
- d) memory
- e) status

2) **behaviour:**

- a) print file
- b) stop printing
- c) remove file from queue

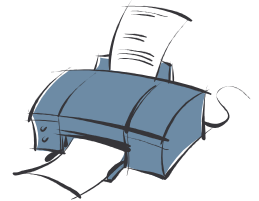
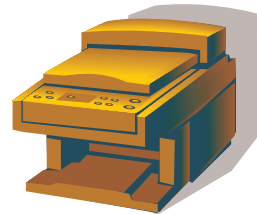
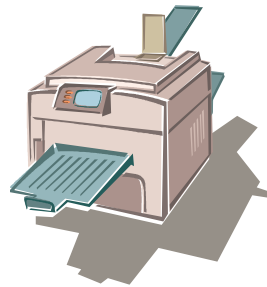
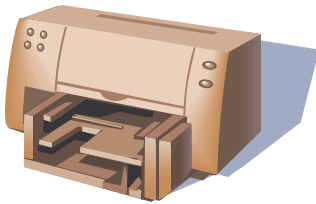
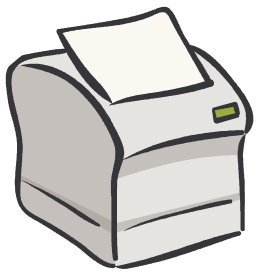
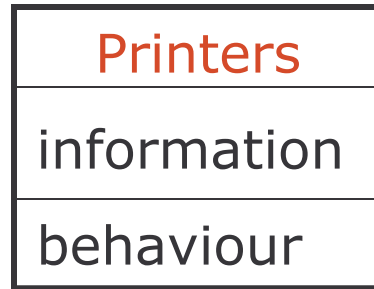
Classes

What is a **class**?

- 1) any uniquely identified abstraction of a set of logically related instances that share the same or similar characteristics
- 2) rules that define objects
- 3) a definition or template that describes how to build an accurate representation of a specific type of objects
- 4) examples: agency, citizen, car, ...

Objects are instantiated (created) using class definitions as templates.

Example: Classes



Attributes

Definition

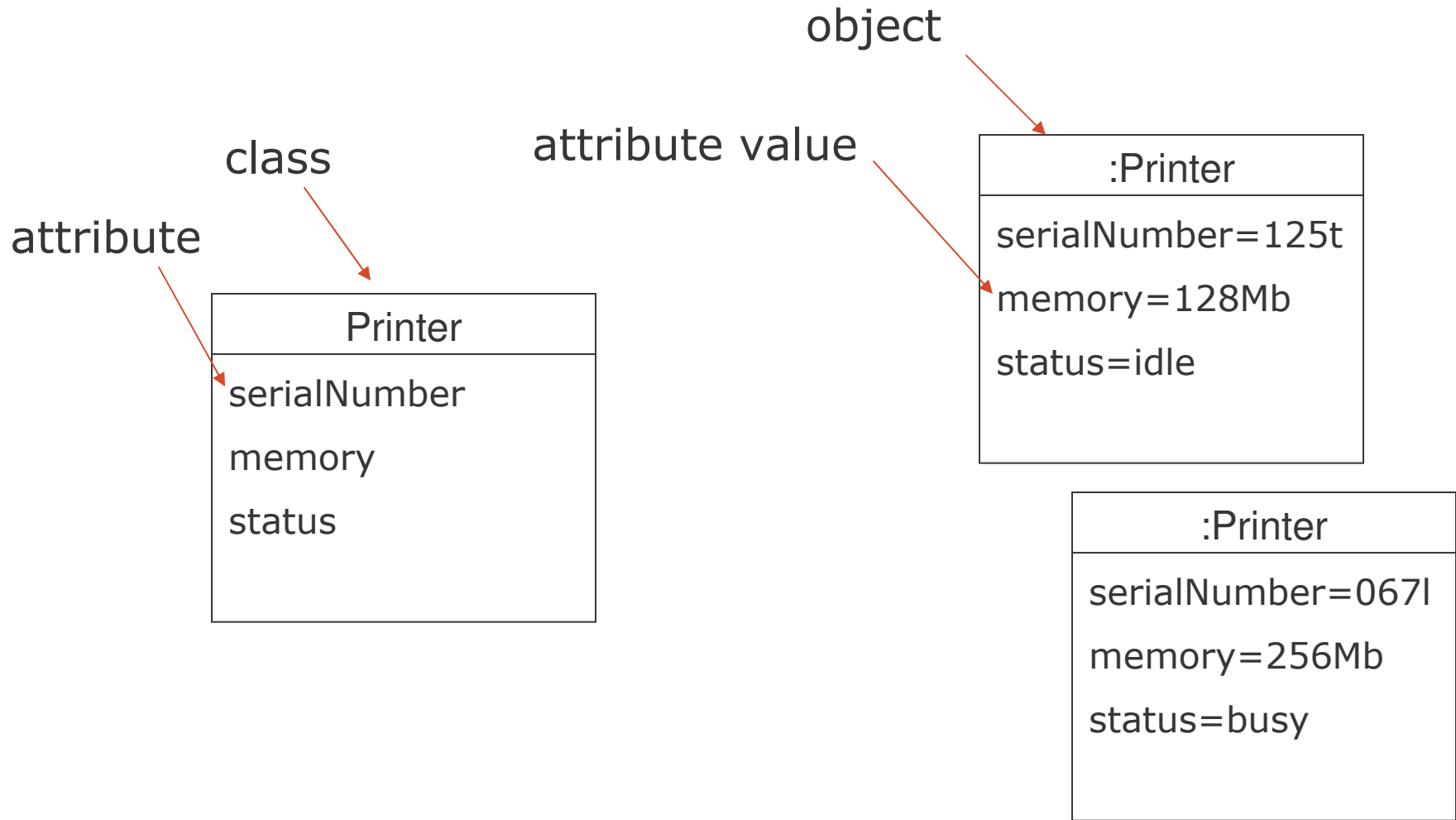
Attribute is a named property of a class that describes a range of values that instances of the class may hold for that property.

An attribute has a type and defines the type of its instances.

Only the object itself should be able to change the values of its attributes.

The given set of values of the attributes defines the state of the object.

Example: Attributes



Operations

Definition

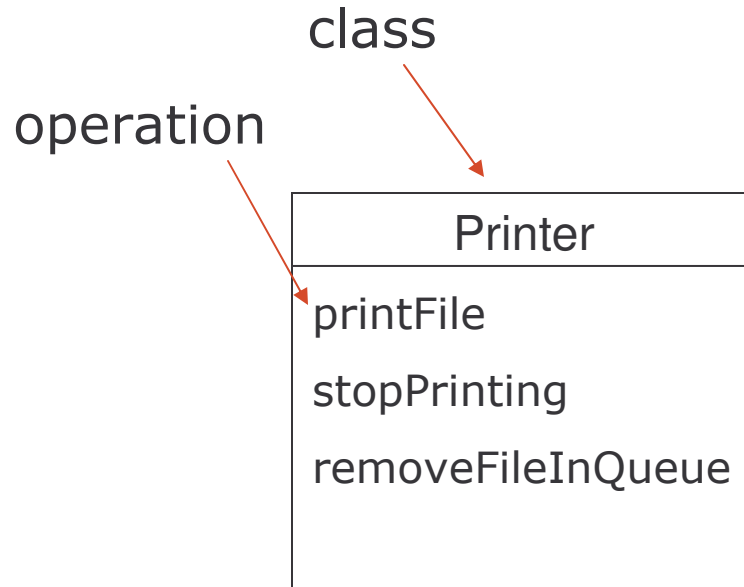
Operation is the implementation of a service that can be requested from any object of the class.

An operation could be:

Question - does not change the values of the object

Command - may change the values of the object

Example: Operations



Applying Abstraction

Abstraction in Object Orientation:

- 1) use of objects and classes in representing reality
- 2) software manages abstractions based on the changes occurring to real-world objects



Courtesy: XML Bible

Encapsulation 2

- 1) hallmark of object orientation
- 2) behaviour is defined once and stored inside objects
- 3) emphasizes two types of information:
 - a) interface information - minimum information for using the object
 - b) implementation information - information required to make an object work properly

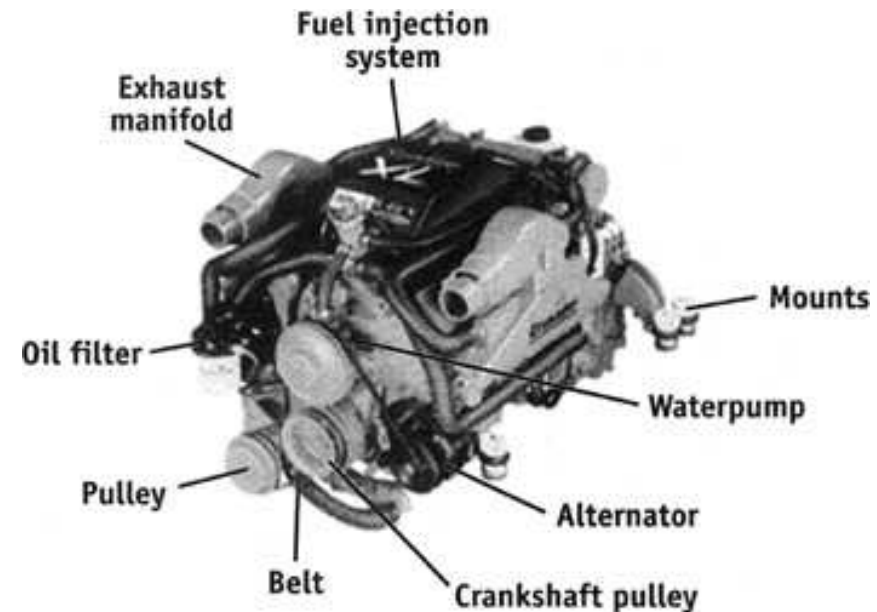
Interface

- 1) minimum information required to use an object
- 2) allows users to access the object's knowledge
- 3) must be exposed
- 4) provides no direct access to object internals



Implementation

- 1) information required to make an object work properly
- 2) a combination of the behaviour and the resources required to satisfy the goal of the behaviour
- 3) ensures the integrity of the information on which the behaviour depends



Encapsulation Requirements

- 1) to expose the purpose of an object
- 2) to expose the interfaces of an object
- 3) to hide the implementation that provides behaviour through interfaces
- 4) to hide the data within an object that defines its structure and supports its behaviour
- 5) to hide the data within an object that tracks its state

Encapsulation Benefits

- separation of an **interface** from **implementation**, so that one interface may have multiple implementations
- data held within one object cannot be corrupted by other objects

Associations and Links

Association:

- expresses relationships between classes
- defines links between instances of classes (objects)

Link:

- 1) expresses relationships between objects

There are different kinds of relationships between classes:

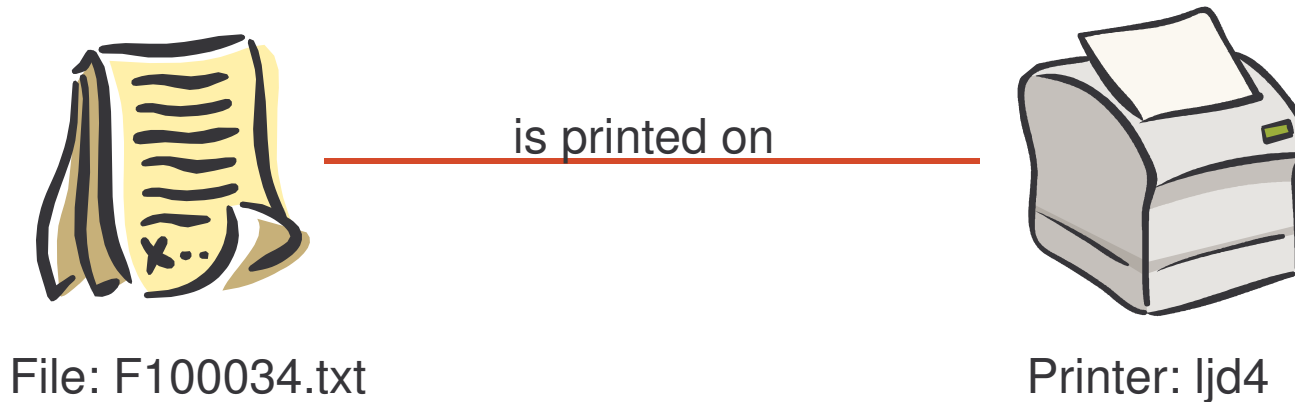
- 1) **association**
- 2) **aggregation**
- 3) **composition**

Example: Associations - Links

a) Association:



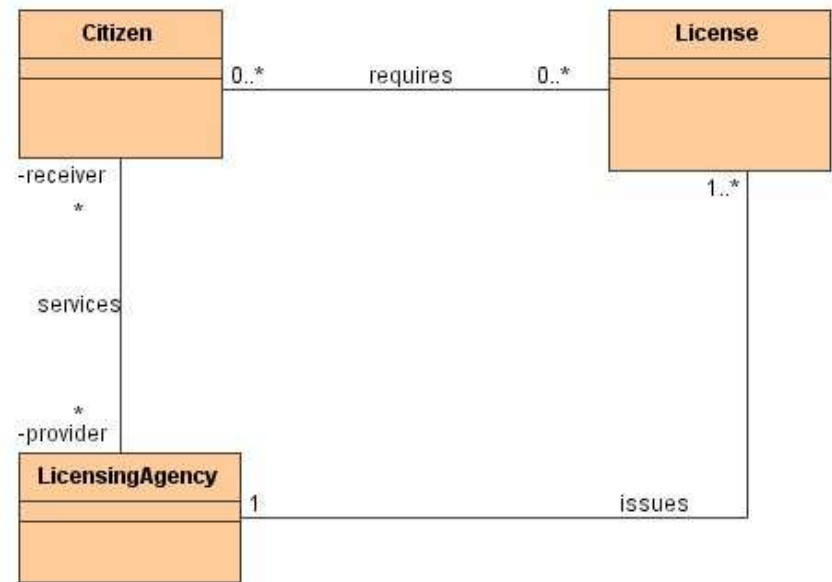
b) Link:



Relationship: Association

Features:

- 1) the simplest form of relationship between classes
- 2) a peer-to-peer relationship
- 3) one object is generally aware of the existence of other object
- 4) implemented in objects as references



Example: Associations

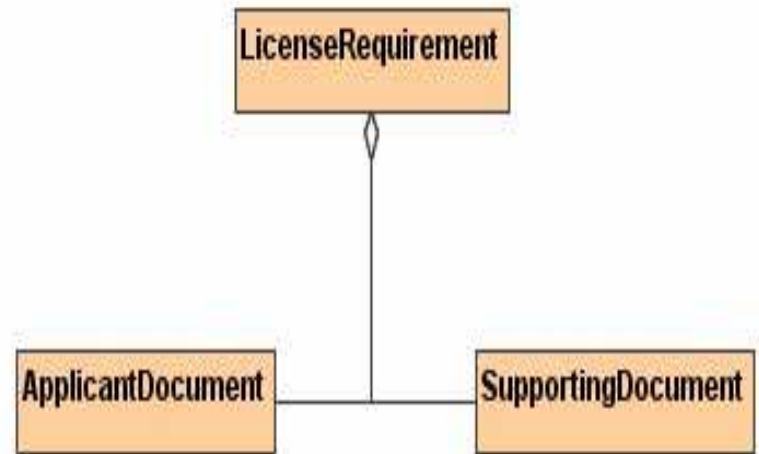
Association between classes A and B:

- 1) A is a physical/logical part of B
- 2) A is a kind of B
- 3) A is contained in B
- 4) A is a description of B
- 5) A is a member of B
- 6) A is an organization subunit of B
- 7) A uses or manages B
- 8) A communicates with B
- 9) A follows B
- 10) A is owned by B
- 11)...

Relationship: Aggregation

Features:

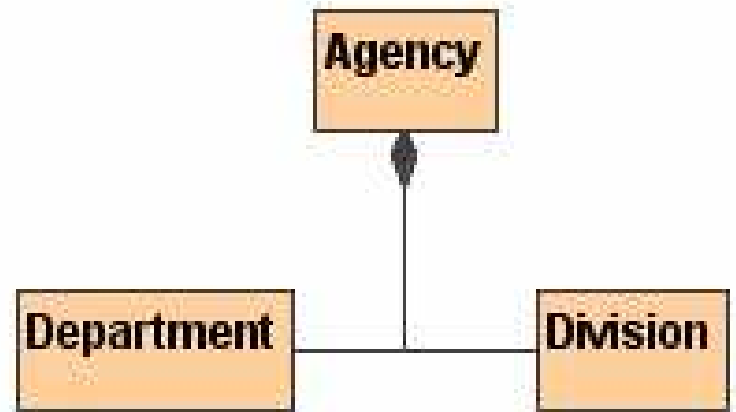
- 1) a more restrictive form of “part-of” association
- 2) objects are assembled to create a new, more complex object
- 3) assembly may be physical or logical
- 4) defines a single point of control for participating objects (parts)
- 5) the aggregate object coordinates its parts



Relationship: Composition

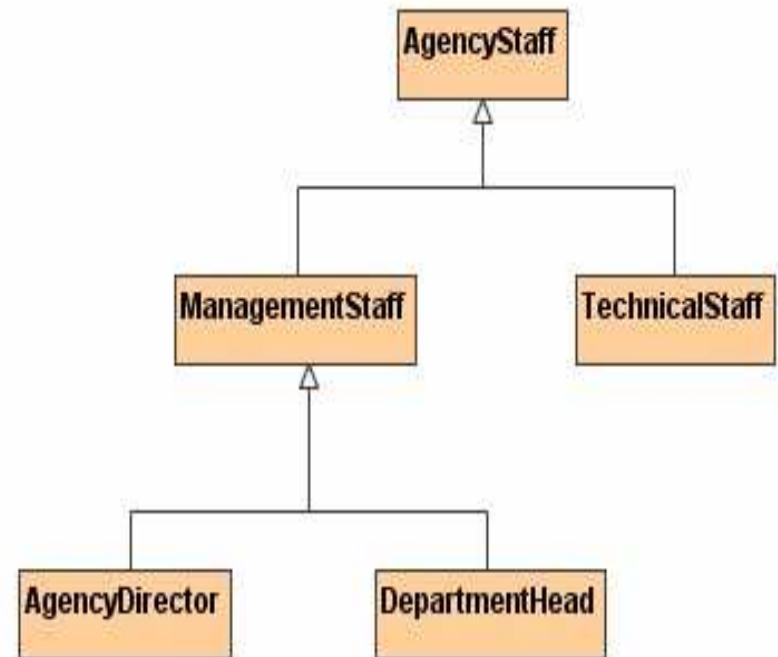
Features:

- 1) a stricter form of aggregation
- 2) lifespan of parts depend on the on lifespan of the aggregate object
- 3) parts cannot exist on their own
- 4) there is a create-delete dependency of the parts on the whole



Inheritance or Generalization

- 1) a process of organizing the features of different kinds of objects that share the same purpose
- 2) equivalent to “kind-of” or “type-of” relationship
- 3) also referred to as inheritance
- 4) specialization is the opposite of generalization
- 5) not an association!



Super-Class and Sub-Class

Definition

Super-class is a class that contains features that are common to two or more classes.

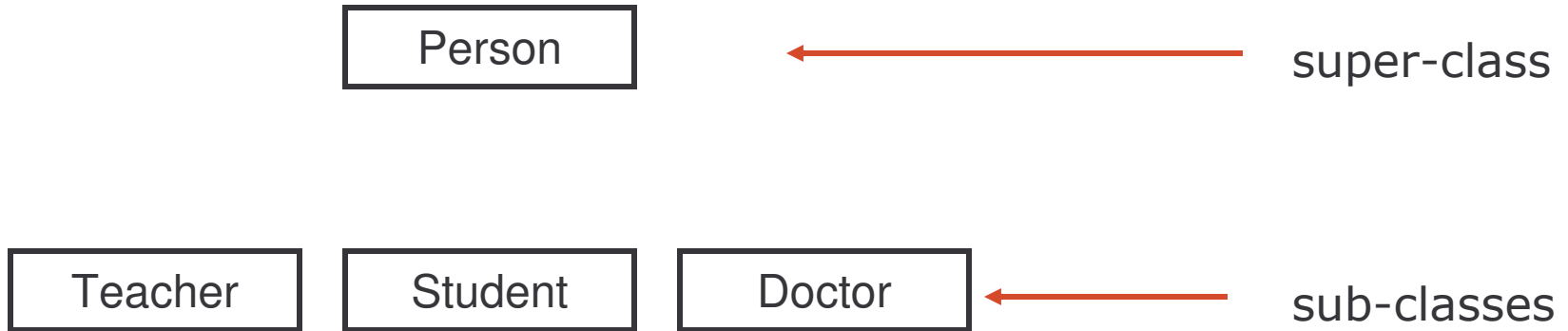
A super-class is similar to a superset. e.g. agency-staff.

Definition

Sub-class is a class that contains features of its super-class or super-classes, and perhaps more.

A class may be a sub-class and a super-class at the same time. e.g. management-staff.

Example: Super- and Sub-Class



Polymorphism

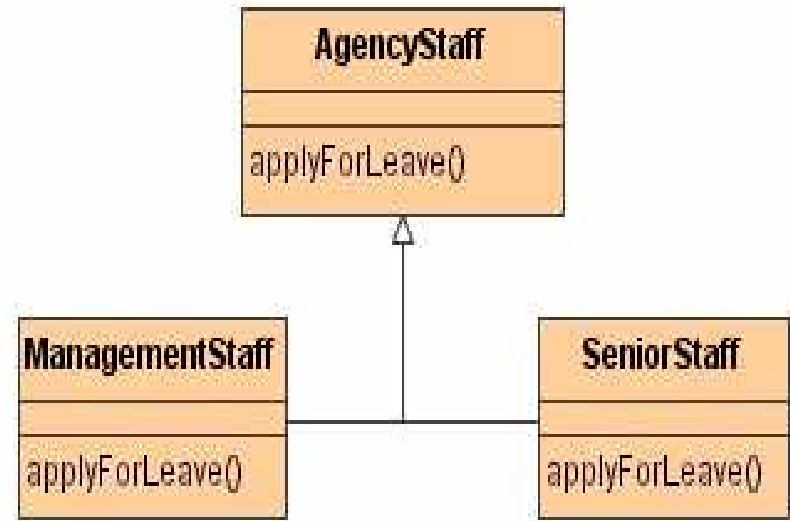
What is it?

- 1) ability to dynamically choose the method for an operation at run-time or service time
- 2) facilitated by encapsulation and generalization:
 - a) **encapsulation**: separation of interface from implementation
 - b) **generalization**: organizing information such that the shared features reside in one class and unique features in another
- 3) therefore: operations could be defined and implemented in a super-class, but the re-implemented methods are in unique sub-classes.

Example: Polymorphism

Many ways of doing the same thing!

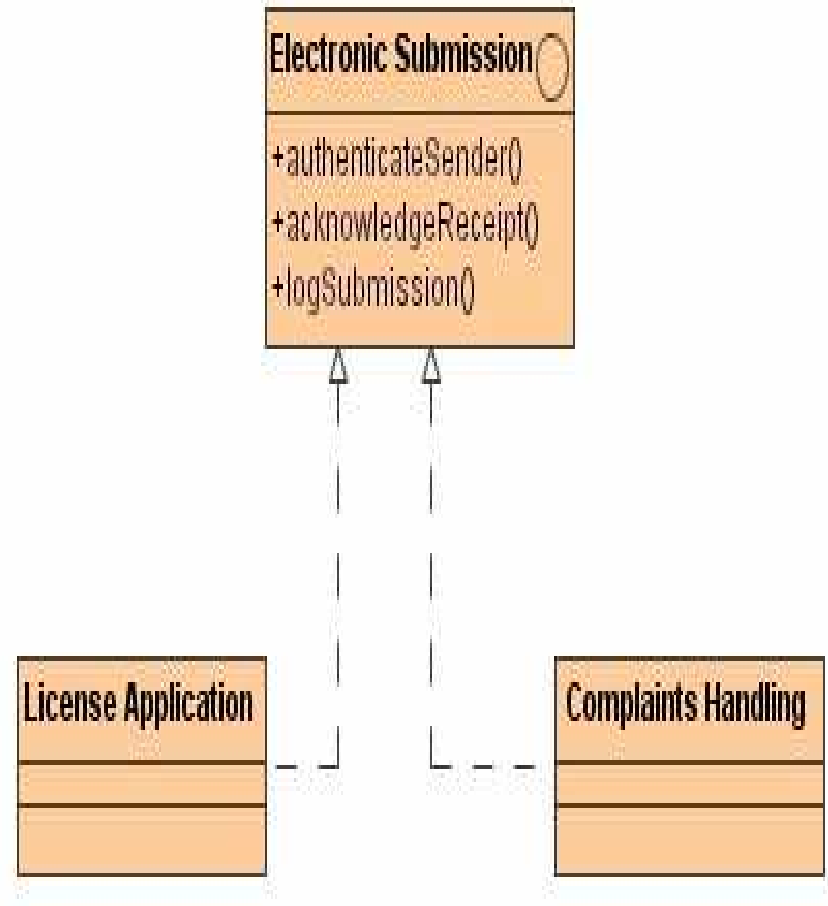
Example: management-staff and agency-staff can apply for leave, but possibly in different ways



Realization

Realization

- 1) allows a class to inherit from an interface class without being a subclass of the interface class
- 2) only inherits operations
- 3) cannot inherit methods, attributes, and associations



Object Oriented Analysis

Object Oriented Analysis 1

- 1) a discovery process
- 2) clarifies and documents the requirements of a system
- 3) focuses on understanding the problem domain
- 4) discovers and documents the key classes in the problem domain
- 5) concerned with developing an object-oriented model of the application domain
- 6) identified objects reflect the entities that are associated with the problem to be solved

Object Oriented Analysis 2

Definition [SEI]

Object Oriented Analysis (OOA) is concerned with developing software requirements and specifications expressed as a system's object model (composed of a population of interacting objects), as opposed to the traditional data or functional views of systems.

Benefits

- 1) **maintainability**: simplified mapping to the real world
 - a) less analysis effort
 - b) less complexity in system design
 - c) easier verification by the user
- 2) **reusability**: reuse of the analysis artifacts that are independent of the analysis method or programming language
- 3) **productivity**: direct mapping to features of OOP languages

Object Oriented Design

Object Oriented Design 1

- process of invention and adaptation
- creates abstraction and mechanisms necessary to meet the system's behavioural requirements determined during analysis
- language independent
- provides an object-oriented model of a software system to implement the identified requirements

Object Oriented Design 2

Definition [SEI]

Object Oriented Design (OOD) is concerned with developing an object-oriented model of a software system to implement the requirements identified during OOA.

The benefits are the same as those in OOA.

Process

Stages in OOD:

- 1) understand and define the context and modes of use of the system
- 2) design the system architecture
- 3) identify the principal objects in the system
- 4) develop design models
- 5) specify object interfaces

Note on OOD activities:

- 1) activities are not strictly linear but interleaved
- 2) back-tracking may be done a number of times due to refinement or availability of more information

Design Principles

Cohesion and coupling

- 1) module attributes
- 2) cohesion - how tightly bound are the internal elements of different modules
- 3) coupling – to what extent one module is connected with others
- 4) for reusability, modules should have **high cohesion** and **low coupling**

Summary

Summary 1

Object is any abstraction that models a single thing in a universe with some properties and behaviour.

A class is any uniquely identified abstraction of a set of logically related objects that share similar characteristics.

Classes may be related by the following types of relationships:

- 1) association
- 2) aggregation
- 3) composition

Summary 2

Object Orientation is characterized by three fundamental principles:

- 1) **encapsulation** – combination of data and behaviour, information hiding, separation of an interface from implementation
- 2) **inheritance** – generalization and specialization of classes, forms of hierarchy
- 3) **polymorphism** – different implementations for the shared operation depending on the particulars of the involving object in the inheritance hierarchy.

Object Oriented Analysis is concerned with creating requirements specifications and analysis models of the application domain.

Object Oriented Design is concerned with implementing the requirements identified during OOA, in the solution domain.

UML Basics

UML Basics

Modelling Principles

UML Basics

1) Modelling Principles

2) UML Overview:

- a) Goals of UML
- b) Brief history of UML
- c) Language architecture

3) Building Blocks:

- a) Elements: Structural, Behavioural, Grouping and Annotation
- b) Relationships

3) Building Blocks:

c) Diagrams:

- Class
- Object
- Use case
- Interaction: Sequence and Collaboration
- Statechart
- Activity
- Component
- Deployment

4) Views

What Is Modelling?

- 1) representation or simplification of reality
- 2) provides a blueprint of a system
- 3) includes elements with broad effects and omits those not relevant at a given level of abstraction

Why Modelling?

- 1) to better understand the system we are developing
- 2) to provide a model of the structure or behaviour of the system
- 3) to experiment by exploring multiple solutions
- 4) to furnish abstraction for managing complexity
- 5) to document the design decisions
- 6) to visualize the system "as-is" and "to-be"
- 7) to provide a template for constructing a system

Why Modelling Graphically?

1) Graphics reveal data

2) 1 bitmap = 1 megaword [anonymous visual modeler]

- Courtesy Cris Kobryn - Introduction to UML

Modelling Principles

- 1) the choice of models has a profound influence on how a problem is attacked and how the solution is shaped
- 2) every model may be expressed at different levels of abstraction (precision)
- 3) effective models are connected to reality
- 4) No single model is sufficient. Non trivial systems are best described with a set of independent but related models

UML Basics

UML Overview

UML Basics

1) Modelling Principles

2) UML Overview:

a) Goals of UML

b) Brief history of UML

c) Language architecture

3) Building Blocks:

a) Elements: Structural, Behavioural, Grouping and Annotation

b) Relationships

3) Building Blocks:

c) Diagrams:

- Class

- Object

- Use case

- Interaction: Sequence and Collaboration

- Statechart

- Activity

- Component

- Deployment

4) Views

What Is The UML?

- 1) UML is a language for visualizing, specifying, constructing and documenting artifacts of software intensive systems.
- 2) Examples of artifacts: requirements, architecture, design, source code, test cases, prototypes, etc.
- 3) UML is suitable for modelling various kinds of systems: enterprise information systems, distributed web-based applications, real-time embedded system, etc.

UML – Specification Language

- 1) provides views for development and deployment
- 2) UML is process independent
- 3) recommended for use with processes that are:
 - a) use-case driven
 - b) architecture-centric
 - c) iterative
 - d) incremental

Goals of UML

- 1) provide modelers with a ready to use, expressive and visual modelling language to develop and exchange meaningful models
- 2) provide extensibility and specialization mechanisms to extend core concepts
- 3) support specifications that are independent of particular programming languages and development processes
- 4) provide a formal basis for understanding the specification language
- 5) encourage/growth of the object tools market
- 6) supports higher level of development with concepts such as components frameworks or patterns

Brief History of UML

- 1) started as a unification of the Booch method and the Rumbaugh method - *Unified Method v. 0.8* (1995), and in 1996 Jacobson joined them to produce UML 0.9
- 2) UML Partners worked with the Amigos to propose UML as a standard modelling language to OMG in 1996.
- 3) in 1997, the UML partners tendered their initial proposal (UML 1.0) to OMG, and 9 months later they submitted the final proposal (UML 1.1)
- 4) minor revision is UML 1.4 adopted in May 2001, and most recent revision is UML 1.5 published in March 2003.
- 5) awaiting UML 2.0 official release.

UML Language Architecture 1

UML language architecture was provided by OMG to align UML with other OMG's technologies.

UML architecture follows the meta-modelling architecture of OMG's Meta-Object Facility (MOF).

Four layers:

- 1) meta-metamodel layer
- 2) metamodel layer
- 3) model
- 4) user objects

UML Language Architecture 2

Layer	Description	Example
Meta-metamodel	The infrastructure for a metamodeling architecture. Defines the language for specifying metamodels.	MetaClass, MetaAttribute, MetaOperation
Metamodel	An instance of a meta-metamodel. Defines the language for specifying a model.	Class, Attribute, Operations, Component
Model	An instance of a metamodel. Defines a language to define an information domain.	Agency, AgencyCode, numberUnits, staffStrength
User object (or user data)	An instance of a model. Defines a specific information model.	<CPSP, 5, 100>

UML Basics

Building Blocks

UML Basics

1) Modelling Principles

2) UML Overview:

- a) Goals of UML
- b) Brief history of UML
- c) Language architecture

3) Building Blocks:

- a) Elements: Structural, Behavioural, Grouping and Annotation
- b) Relationships

3) Building Blocks:

c) Diagrams:

- Class
- Object
- Use case
- Interaction: Sequence and Collaboration
- Statechart
- Activity
- Component
- Deployment

4) Views

UML Building Blocks

Three basic building blocks:

- 1) **elements**: main “citizens” of the model
- 2) **relationships**: relationships that tie elements together
- 3) **diagrams**: mechanisms to group interesting collections of elements and relationships

These building blocks will be used to represent large and small complex structures.

Elements

Four basic types of elements:

- 1) structural
- 2) behavioural
- 3) grouping
- 4) annotation

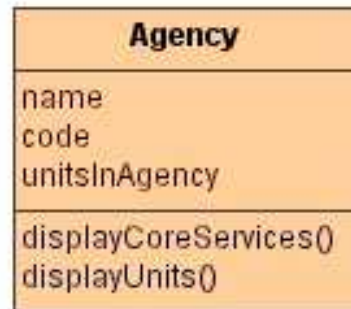
They will be used to specify well-formed models.

Structural Elements

- 1) static part of the model to represent conceptual or physical elements
- 2) “nouns” of the model
- 3) seven kinds of structural elements:
 - a) class
 - b) interface
 - c) collaboration
 - d) active class
 - e) use case
 - f) component
 - g) node

Class 1

- 1) description of a set of objects that share the same attributes, operations, relationships and semantics
- 2) implements one or more interfaces
- 3) graphically rendered as a rectangle usually including a name, attributes and operations



- 4) can be also used to represent actors, signals and utilities

Interface

- 1) collection of operations that specifies a service of a class
- 1) describes the externally visible behaviour (partial or complete) of a class
- 2) defines a set of operation signatures but not their implementations
- 3) rendered as a circle with a name.



Collaboration

- 1) defines an interaction between elements
- 2) several elements cooperating to deliver a behaviour rather than individual behaviour
- 3) includes structural and behavioural dimensions
- 4) represents implementations of patterns that make up a system
- 5) represented as a named ellipse drawn with a dashed line



Use Case

- 1) description of a sequence of actions that a system performs to deliver an observable result to a particular actor
- 2) used to structure the behavioural elements in a model
- 3) realized by collaboration
- 4) graphically rendered as an ellipse drawn with a solid line



Active Class

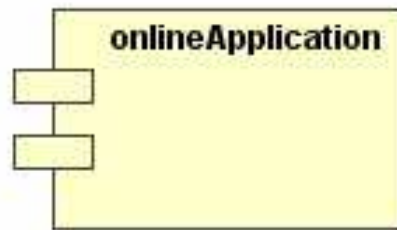
- 1) a class whose objects own one or more processes or threads and therefore can initiate an action
- 2) class whose objects have concurrent behaviour with other objects
- 3) graphically, an active class is rendered just like a class drawn with a thick line



- 4) it also can be used to represent processes and threads

Component

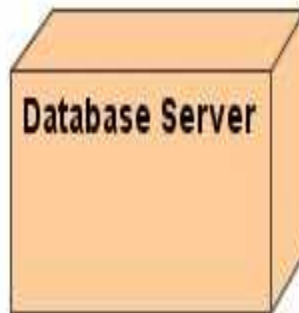
- 1) physical replaceable part of a system that conforms to and provides the realization of a set of interfaces
- 2) represents deployment components such as COM+ or Java Beans components
- 3) represents a physical packaging of logical elements such as classes, interfaces and collaborations



- 4) it also can be used to represent applications, files, libraries, pages and tables.

Node

- 1) a physical element that exists at run time
- 2) represents a computational resource with memory and processing capacity
- 3) a set of components may reside in a node
- 4) components may also migrate from one node to another
- 5) graphically modelled as a cube.



Behavioural Elements

- 1) represent behaviour over time and space
- 2) “verbs” of the model
- 3) two kinds of behavioural elements:
 - a) interaction
 - b) state machine

Interaction

- 1) a set of messages exchanged among a set of objects within a particular context to accomplish a specific purpose
- 2) specifies the behaviour of a set of objects
- 3) involves a number of other elements:
 - messages, action sequences (behaviour invoked by a message) and links (connection between objects)
- 4) graphically rendered as an arrow

saveapplication()
→

State Machine

- 1) specifies a sequence of states an object or an interaction goes through during its lifetime and its response to external events
- 2) may specify the behaviour of an individual class or a collaboration of classes
- 3) includes a number of elements including states, transition, events and activities
- 4) presented as a rounded rectangle with a name and sub-states



- 5) interactions and state machines are associated with structural elements such as classes, collaborations, and objects

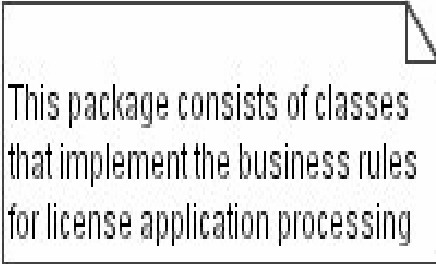
Grouping Elements – Packages

- 1) organizational part of UML
- 2) the primary mechanism for grouping and decomposition
- 3) purely conceptual and only available at development time
- 4) graphically represented as a tabbed folder



Annotation Elements – Notes

- 1) comments added to models for better explanation or illumination on specific elements
- 2) explanatory aspect of UML models
- 3) notes are used primarily for annotation e.g. for rendering constraints and comments attached to elements or collections of elements
- 4) presented as a rectangle with a dog-eared corner
- 5) may include both textual and graphical comments



This package consists of classes that implement the business rules for license application processing

Relationships

Four basic types of relationships:

- 1) dependency
- 2) associations
- 3) generalization
- 4) realization

Meanings are consistent with the basic OO relationship types described earlier

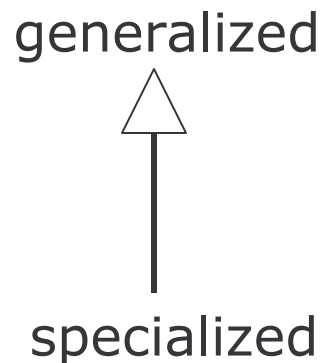
Relationship: Association

- 1) a structural relationship describing a set of links
- 2) links are connections between objects
- 3) aggregation is a special type of association depicting whole-part relationship
- 4) association is presented as a solid line, possibly directed, labelled and with adornments (multiplicity and role names)



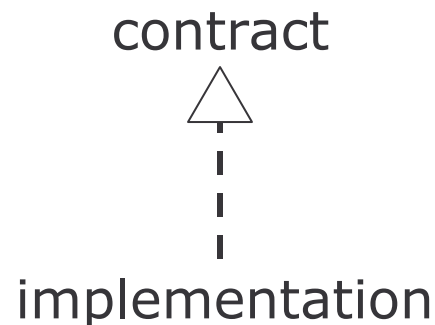
Relationship: Generalization

- 1) a relationship in which objects of a specialized element (child) are substitutable for objects of a generalized element (parent)
- 2) child elements share the structure and behaviour of the parent
- 3) rendered graphically as a solid line with hollow arrowhead pointing to the parent



Relationship: Realization

- 1) a semantic relationship between elements, wherein one element specifies a contract and another guarantees to carry out this contract
- 2) relevant in two basic scenarios:
 - a) interfaces versus realizing classes or components
 - b) uses cases versus realizing collaborations
- 3) graphically depicted as a dashed arrow with hollow head a cross between dependency and generalization



Variations to Relationships

Variations of these four relationship types include:

1) refinement

2) trace

3) include

4) extend

Diagrams

- 1) a graph presentation of a set of elements and relationships where:
 - a) nodes are elements
 - b) edges are relationships

- 2) can visualize a system from various perspective thus a projection of a system

- 3) UML is characterized by nine major diagrams: a) class, b) object, c) use case, d) sequence, e) collaboration, f) statechart, g) activity, h) component and i) deployment.

Class and Object Diagrams

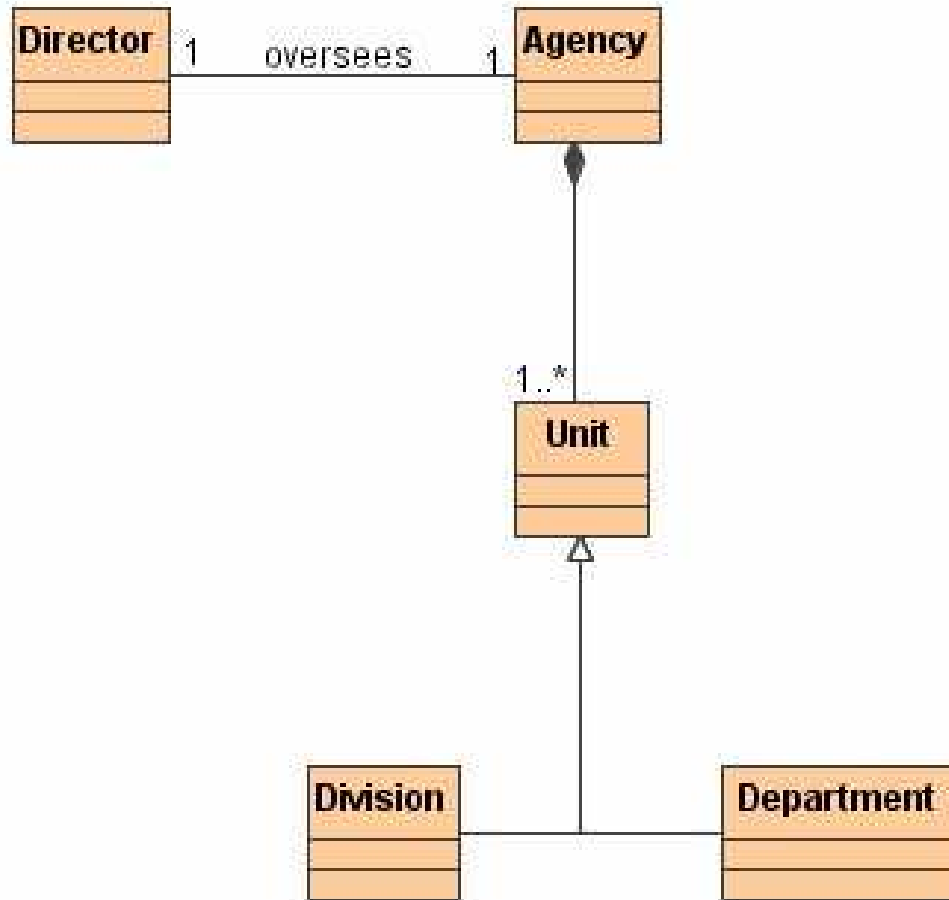
Class Diagrams:

- 1) show a set of classes, interfaces and collaborations, and their relationships
- 2) address static design view of a system

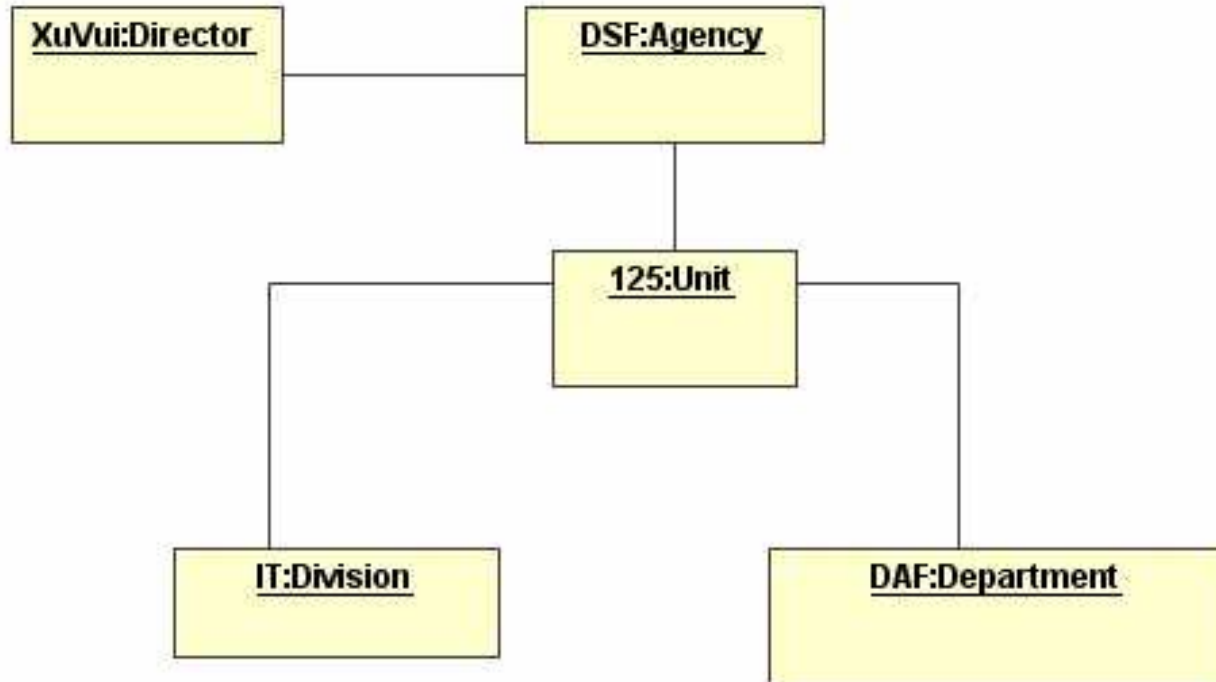
Object Diagrams:

- 1) show a set of objects and their relationships
- 2) static snapshots of element instances found in class diagrams

Example: Class Diagram



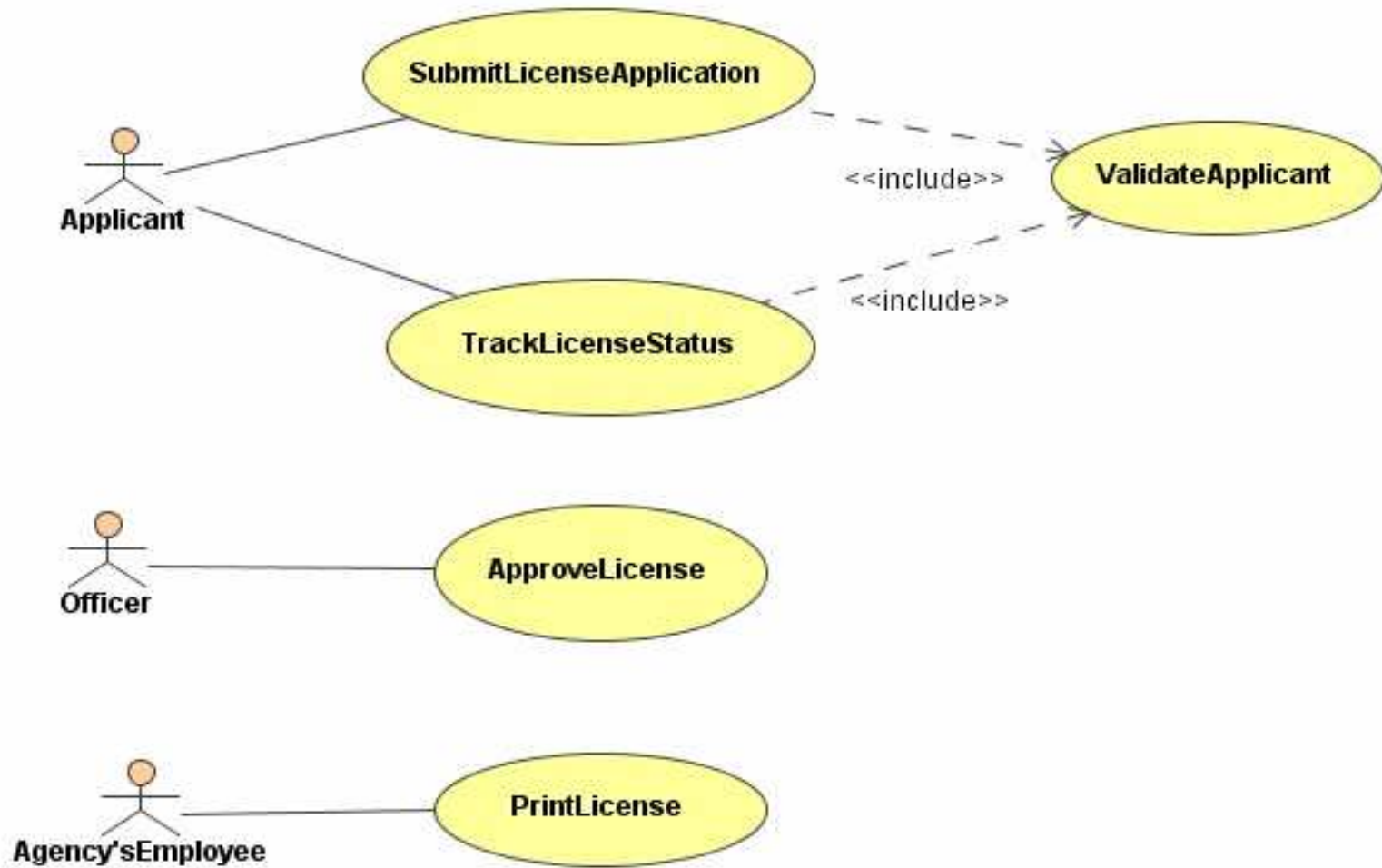
Example: Object Diagram



Use Case Diagrams

- 1) show a set of actors and use cases, and their relationships
- 2) addresses static use case view of the system
- 3) important for organizing and modelling the external behaviour of the system

Example: Use Case Diagram



Interaction Diagrams

Sequence Diagrams:

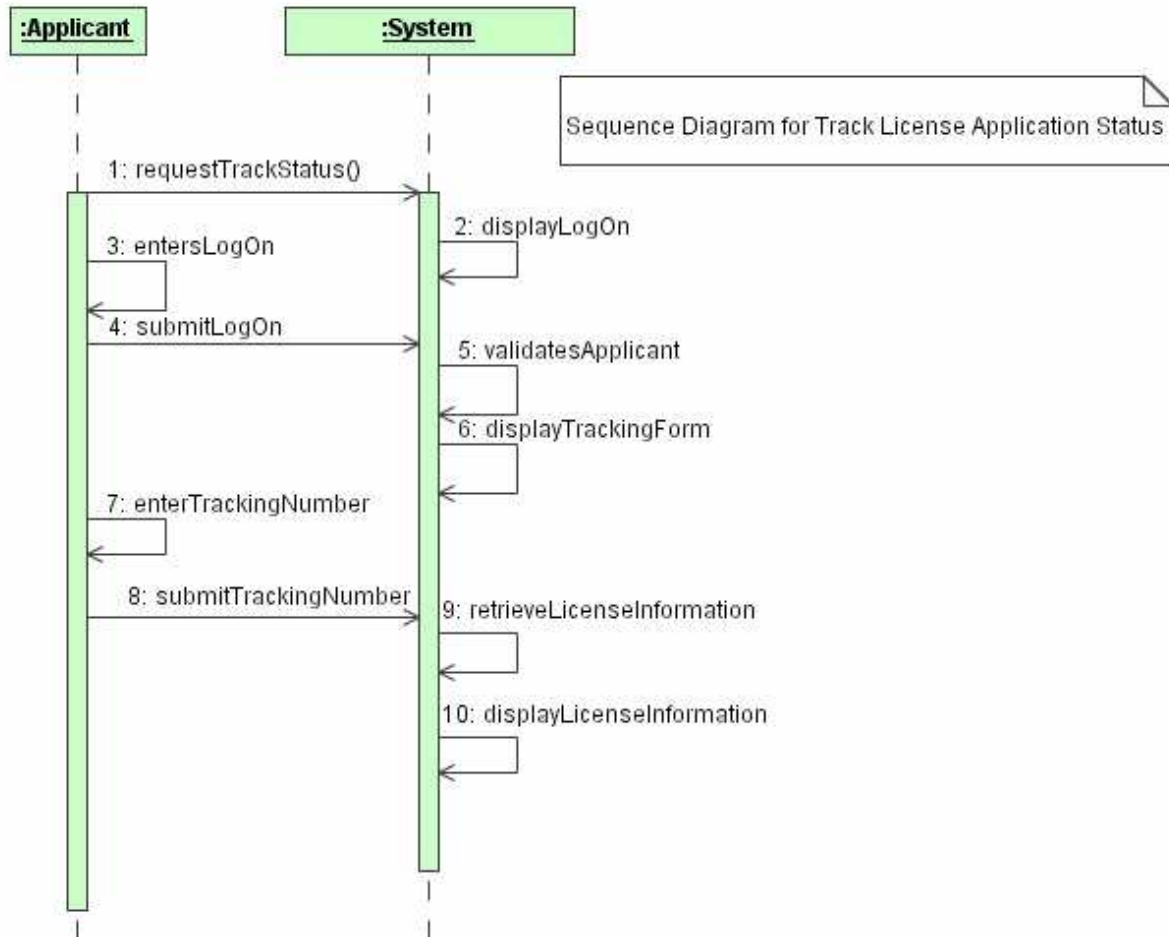
- 1) show interactions consisting of a set of objects and the messages sent and received by those objects
- 2) address the dynamic behaviour of a system with special emphasis on the chronological ordering of messages

Collaboration Diagrams:

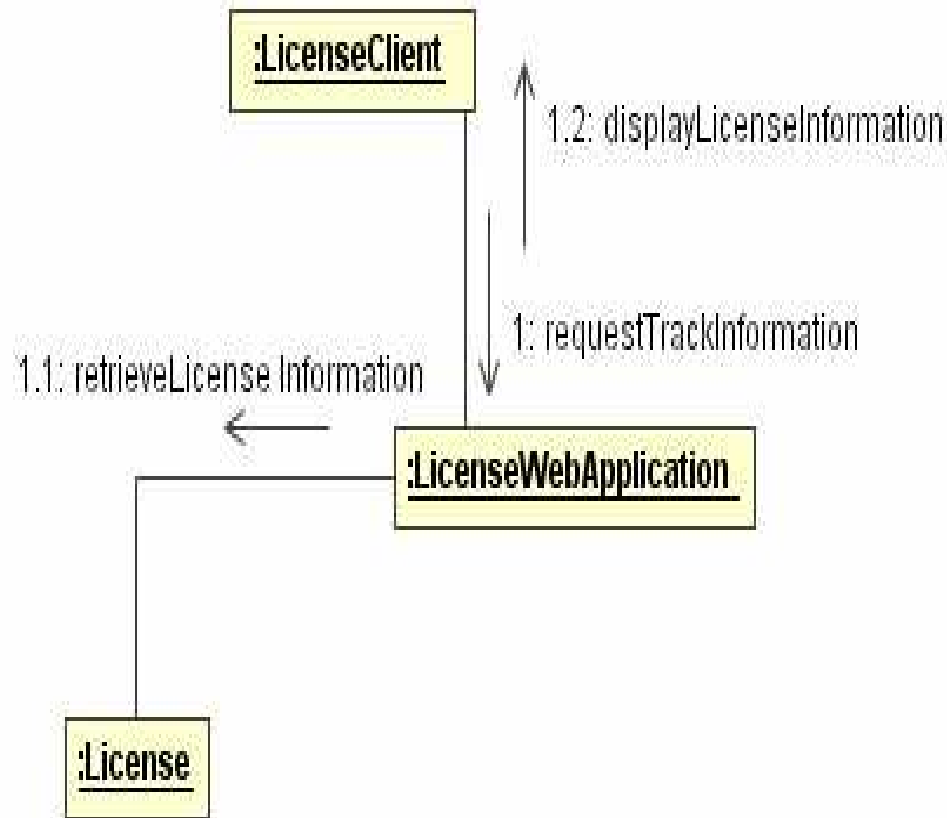
- 1) show the structural organization of objects that send and receive messages

Sequence and collaboration diagram are isomorphic i.e. one can be transformed into another

Example: Sequence Diagram



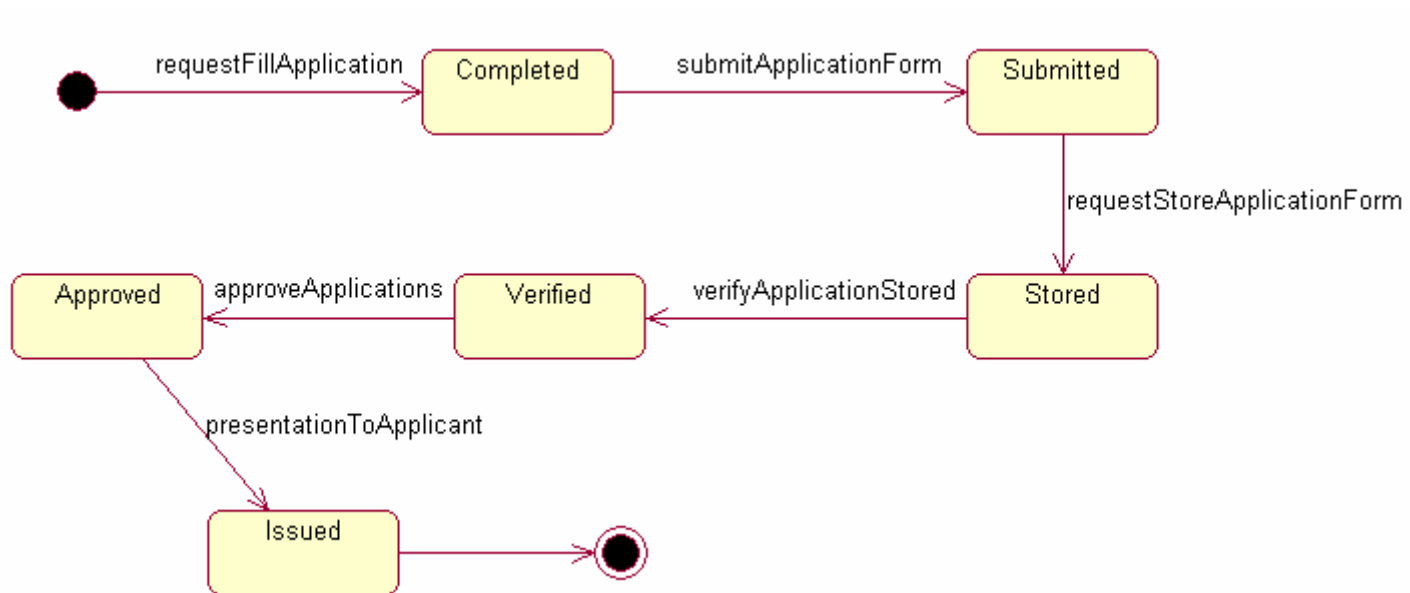
Example: Collaboration Diagram



Statechart Diagrams

- 1) show a state machine consisting of states, transitions, events, and activities
- 2) address the dynamic view of a system
- 3) important in modelling the behaviour of an interface, class or collaboration
- 4) emphasise on event-driven ordering

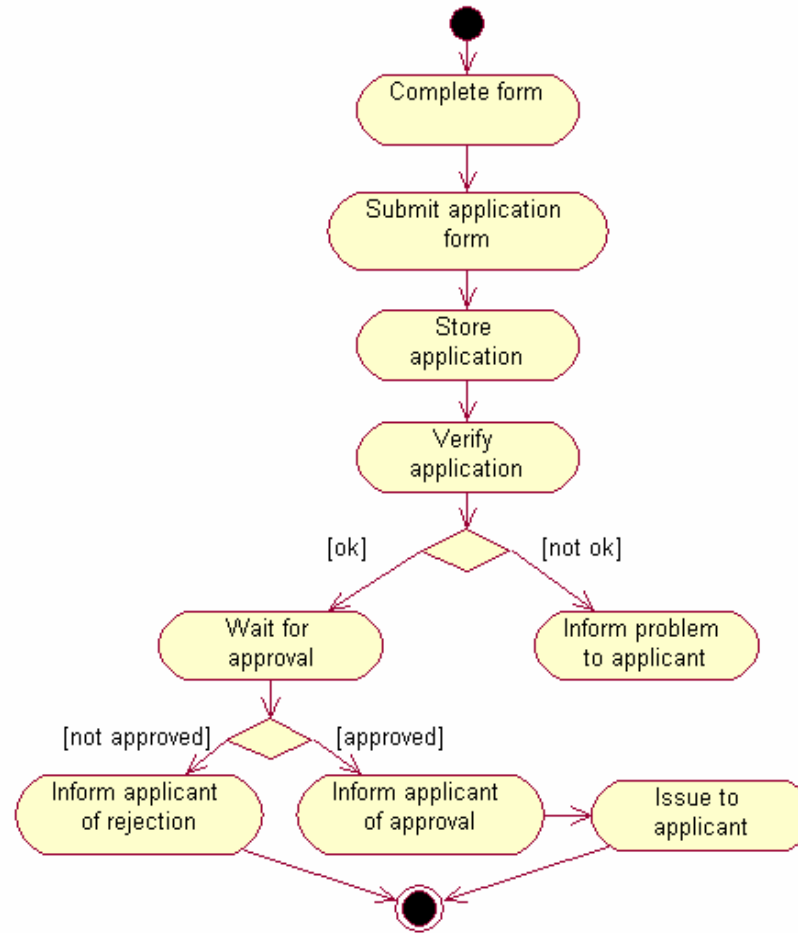
Example: Statechart Diagram



Activity Diagrams

- 1) a diagram showing control/data flows from one activity to another
- 2) addresses the dynamic view of a system, useful for modelling its functions
- 3) emphasises the flow of control among objects

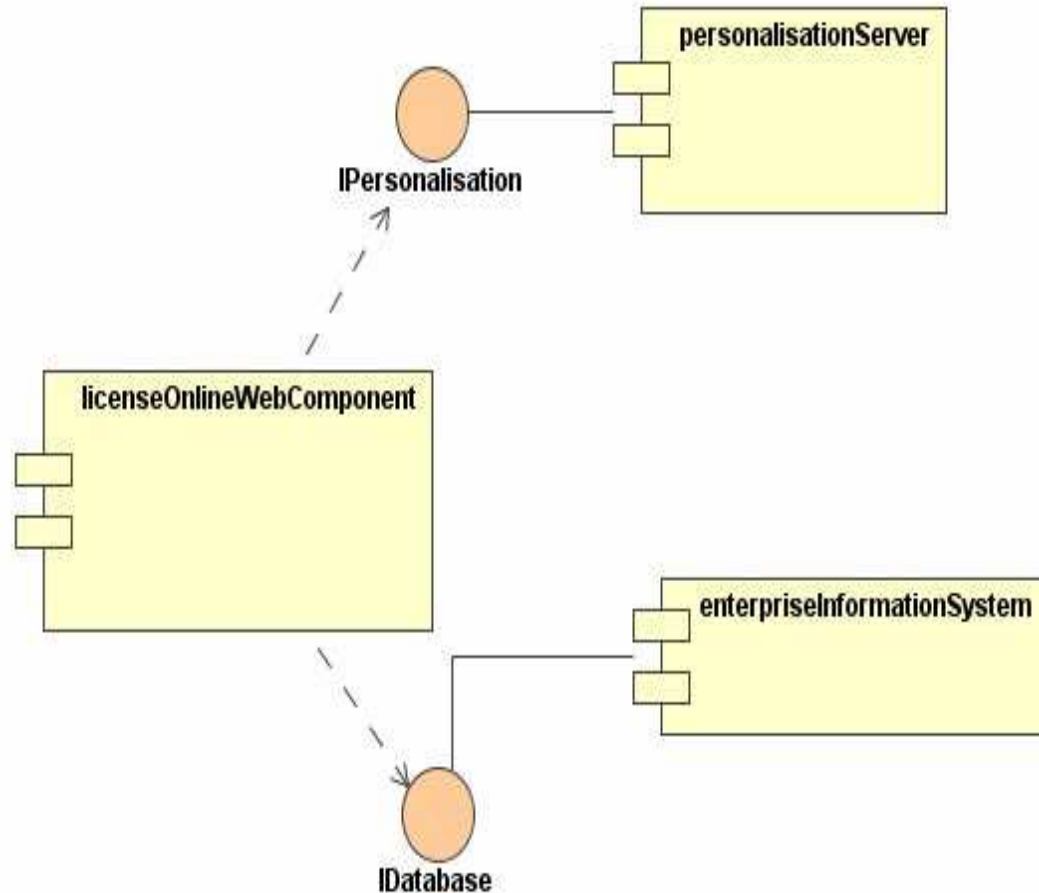
Example: Activity Diagram



Component Diagrams

- 1) show the organization and dependencies amongst a set of components
- 2) address static implementation view of a system
- 3) a component typically maps to one or more classes, interfaces or collaborations

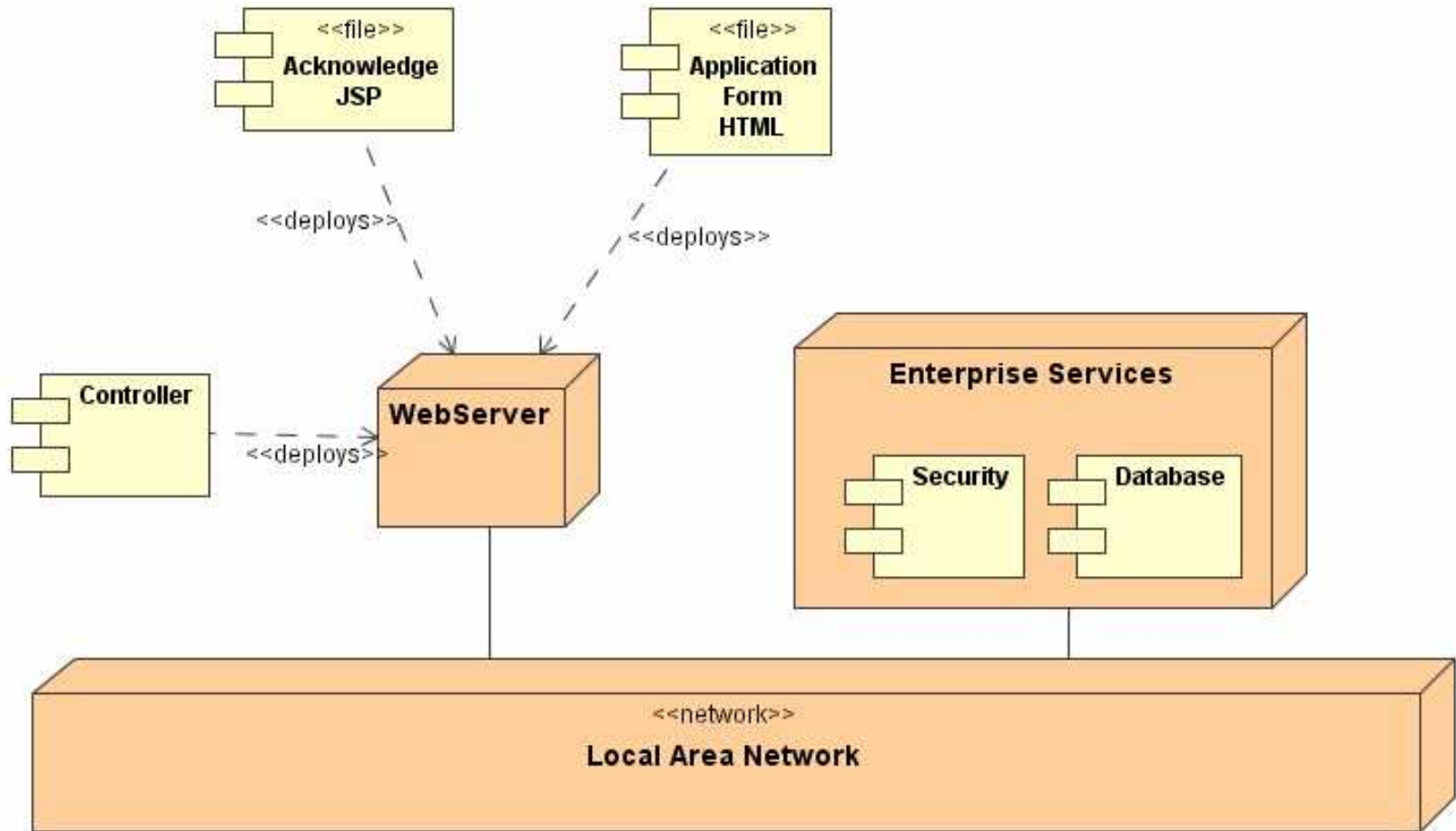
Example: Component Diagram



Deployment Diagrams

- 1) show configuration of run-time processing nodes and the components hosted on them
- 2) address the static deployment view of an architecture
- 3) related to component diagrams with nodes hosting one or more components

Example: Deployment Diagram



UML Basics

Views

UML Basics

1) Modelling Principles

2) UML Overview:

- a) Goals of UML
- b) Brief history of UML
- c) Language architecture

3) Building Blocks:

- a) Elements: Structural, Behavioural, Grouping and Annotation
- b) Relationships

3) Building Blocks:

c) Diagrams:

- Class
- Object
- Use case
- Interaction: Sequence and Collaboration
- Statechart
- Activity
- Component
- Deployment

4) Views

Modelling Views 1

- 1) **Use case view**: describes the behaviour of the system as seen by its end users, analysts and testers. This view shapes the system architecture.
- 2) **Design view**: encompasses the classes, interfaces, interfaces, and collaborations that form the vocabulary of the problem and its solution.
- 3) **Process view**: encompasses the threads and processes that form the system's concurrency and synchronization mechanisms. This view addresses the performance, scalability and throughput of the system.

Modelling Views 2

- 4) **Implementation View**: encompasses the components and files that are used to assemble and release the physical system. This view addresses the configuration management of the system's releases.

- 5) **Deployment View**: encompasses the nodes that form the system's hardware topology on which the system executes.

Modelling Monolithic Systems

- 1) **Use case view:** use case diagrams
- 2) **Design views:** class diagrams (structural modelling) and interaction diagrams (behavioural)
- 3) **Process View:** none
- 4) **Implementation view:** none
- 5) **Deployment view:** none

Modelling Distributed Systems

- 1) **Use case view**: use case diagrams and activity diagram (behavioural modelling)
- 2) **Design views**: class diagrams (structural modelling) interaction diagrams (behavioral modelling), statechart diagram (behavioural)
- 3) **Process View**: class diagram (structural modelling) and interaction diagrams (behavioural)
- 4) **Implementation view**: component diagrams
- 5) **Deployment view**: deployment diagrams

Summary 1

A model provides a blueprint of a system.

UML is a language for visualizing, specifying, constructing and documenting artifacts of software intensive systems.

UML supports five basic views of a system: user, static-structural, dynamic, implementation and environmental modelling views.

UML is process independent but recommended for use with processes that are: use case driven, architecture-centric, iterative and incremental.

Summary 2

There are three building blocks which characterize UML – elements, relationships and diagrams.

Categories of elements in UML include: structural, behavioural, grouping and annotation.

There are four basic types of relationships in UML: dependency, association, generalization and realization.

UML provides 9 diagrams for modelling: class, object, use case, sequence, collaboration, statechart, activity, component and deployment.

There are five different modelling views in UML: use case, design, process, implementation and deployment.