

# Object-Oriented Analysis and Design with UML

## Training Course

---

Adegboyega Ojo  
Elsa Estevez

---

e-Macao Report 19

Version 1.0, October 2005





## Table of Contents

1. Overview .....	1
2. Objectives .....	1
3. Prerequisites .....	2
4. Methodology .....	2
5. Content .....	2
5.1. Introduction .....	2
5.2. Object-Orientation .....	2
5.3. UML Basics .....	3
5.4. UML Modelling .....	3
5.4.1. Requirements .....	3
5.4.2. Architecture .....	3
5.4.3. Design .....	3
5.4.4. Implementation .....	4
5.4.5. Deployment .....	4
5.5. Unified Process .....	4
5.6. Tools .....	4
5.7. Summary .....	4
6. Assessment .....	4
7. Organization .....	4
References .....	6
Appendix .....	7
A. Slides .....	7
A.1. Introduction .....	7
A.2. Object Orientation .....	12
A.2.1. Background .....	12
A.2.3. Concepts .....	17
A.2.4. Object Oriented Analysis .....	28
A.2.5. Object Oriented Design .....	30
A.3. UML Basics .....	33
A.3.1. UML Background .....	33
A.3.2. UML Building Blocks .....	36
A.3.3. Modelling Views .....	49
A.4. Requirements .....	53
A.4.1. Software Requirements .....	53
A.4.2. Use Case Modelling .....	58
A.4.3. Conceptual Modelling .....	66
A.4.4. Behavioural Modelling .....	76
A.5. Architecture .....	94
A.5.1. Software Architecture .....	94
A.5.2. Collaboration Diagrams .....	100
A.5.3. Component Diagrams .....	106
A.5.4. Packages .....	111
A.5.5. Frameworks and Patterns .....	115
A.6. Design .....	123
A.6.1. Software Design .....	123
A.6.2. Design Class Diagrams .....	127
A.6.3. Activity Diagrams .....	134
A.6.4. Sequence Diagrams .....	141
A.6.5. Statechart Diagrams .....	146
A.6.6. Design Patterns .....	153

---

A.7. Implementation .....	159
A.8. Deployment.....	163
A.9. Unified Process .....	167
A.10. UML Tools.....	174
A.11. Summary .....	178
B. Assessment .....	186
B.1. Set 1 .....	186
B.2. Set 2 .....	189

## 1. Overview

The Unified Modelling Language (UML) is a graphical language for visualising, specifying, constructing, and documenting artefacts of software intensive systems [6]. UML represents the unification of a number of efforts to build notations for expressing models of Object Oriented Analysis and Design (OOAD) under the auspices of the Object Management Group (OMG). At present, UML is the de-facto standard for Object Oriented modelling.

This document describes the course "Object Oriented Analysis and Design with UML" taught to the Core and Extended Teams in the context of the e-Macao Project. The course presents the method of Object Oriented Analysis and Design (OOAD) using the UML notation.

Following the introduction of basic concepts and principles of object orientation, the course shows how informal requirements can be described in details. Considering four views of a typical system - user, static, dynamic and implementation, each of the nine UML diagrams are discussed extensively for modelling these views. For instance, the course describes: (i) use cases for modelling requirements, (ii) class and object diagrams for obtaining good understanding of an application domain and (iii) sequence, collaboration and statechart diagrams for analyzing requirements and specifying architectural and design decisions. In addition, the course teaches the best practices in OOAD based on architectural and design patterns and how UML can be used in the context of the Unified Process (UP). It concludes with a comparative analysis of some popular UML tools.

The rest of this document is as follows. Sections 2, 3 and 4 explain respectively the objectives, prerequisites and methodology for teaching the course. The content of the course is introduced in Section 5. The assessment and organization of the course are explained in Sections 6 and 7. Following references, Appendix A includes the complete set of slides and Appendix B contains two sets of assessment questions with answers.

## 2. Objectives

The objectives of the course are as follows:

- 1) Teaching the basic concepts and principles of Object Orientation including Object Oriented Analysis and Design.
- 2) Introducing the Syntax, Semantics and Pragmatics of the Unified Modeling Language.
- 3) Showing how requirements can be described informally and how they are modeled using Use Case Diagrams.
- 4) Teaching how the structural and behavioral aspects of a system can be analyzed, specified and designed using Class, Interaction and Statechart Diagrams.
- 5) Showing how implementation and deployment details of a system can be modeled using the Implementation and Deployment Diagrams respectively.
- 6) Teaching best practices in OOAD based on Architecture and Design patterns.
- 7) Introducing the Unified Process and showing how UML can be used within the process.
- 8) Presenting a comparison of the major UML tools for industrial-strength development.

### 3. Prerequisites

This course assumes that the learner possesses the basic knowledge of application development. No prior knowledge of Object Orientation is required.

### 4. Methodology

The course has been designed based on the following didactic principles:

- *Depth versus Breadth* - As a foundation, an attempt has been made to cover the various aspects of UML as required in practical applications, without much loss of depth.
- *Academic Orientation* - A body of concepts is defined rigorously and incrementally to establish a foundation for proper understanding and use of technology.
- *From Definitions to Demonstrations* - All major concepts introduced during the course are illustrated with small-size examples which are also demonstrated on the computer, whenever possible.
- *From Demonstrations to Assignments* - On the basis of demonstrations, students are asked to perform different tasks with increasing level of difficulty and independence.

Generally, instructors are expected to administer the course material in a tutorial style. Unlike most courses or materials available on UML, this course teaches UML from an application perspective rather than the usual notation perspective.

### 5. Content

The course is organized into six major sections: Object-Orientation, UML Basics, UML Modelling, Unified Process, UML Tools and Summary. The synopses for these sections are presented below. The course starts with an Introduction section, which describes the contents presented in subsequent sections, including the organizational aspect of the course. The course consists of 618 slides divided into six major sections: Introduction, Object Orientation, UML Basics, UML Modelling, Unified Process and UML Tools.

#### 5.1. Introduction

The Introduction section provides information on the aims of the course, the outline of each section, and the schedule for the course as delivered within the context of the e-Macao Project. This part is covered in the slides 1 through 16.

#### 5.2. Object-Orientation

This section is covered in the slides 17 through 83. It presents the background, principles and fundamental concepts of Object Orientation: Object, Class, Abstraction, Interface, Implementation, Aggregation, Composition, Generalization, Sub-Class and Polymorphism. The section concludes with the presentation of Object Oriented Analysis and Design.

### **5.3. UML Basics**

The section provides a brief presentation of UML. It starts by explaining what modelling entails and why it is important for software and system engineering. Next, it outlines the basic building blocks of UML: Elements, Relationships and Diagrams. Finally, the section presents five basic modelling views supported by UML: Use Case, Design, Process, Implementation and Deployment. This section is covered in the slides 84 through 151.

### **5.4. UML Modelling**

The section constitutes the major part of the course. It presents the nine UML diagrams and how they are applied through the five modelling activities: Requirements, Architecture, Design, Implementation and Deployment. The following sections are devoted to each of these modelling activities.

#### **5.4.1. Requirements**

This section teaches about requirements in general. It shows how requirements can be described informally and how they can be modelled as Use Cases. In addition, the section presents how the basic concepts and entities of an application domain can be described using Class and Object Diagrams. Furthermore it teaches how Interaction and Statechart Diagrams can be used to specify class and object interactions and their internal behaviours respectively. This Section is covered in the slides 152 through 299.

#### **5.4.2. Architecture**

The section starts by introducing the various concepts related to Architectural Design: Sub-Systems, Services, Coupling, Cohesion and Layering. Next, it presents Collaborations and how they can be used to model the functional units of a system. Static and dynamic aspects of collaborations are covered as well. Collaboration Diagrams are presented as the major diagram for architecture modelling. Packages are introduced for organizing or modularizing the functional units of a system. The section ends with a discussion of the set of major architectural patterns: Repository, Model-View-Controller, Client-Server, Peer-to-Peer and Pipe-and-Filter. This section is presented in the slides 300 through 403.

#### **5.4.3. Design**

The section, covered in the slides 404 through 526, starts with an introduction to System and Object Design activities. Next, it presents Design Classes as foundation of the whole development. The steps involved in building Design Class Diagrams are described, from reviewing conceptual classes, to decorating design class attributes with data types, to providing default values and constraints. Furthermore, Activity Diagrams are presented for describing business processes, procedures or algorithms. Sequence and Statechart Diagrams are revisited in the discussion of detailed design of both internal and external behaviours of system objects. The section is concluded with a discussion on Design Patterns.

#### 5.4.4. Implementation

This section presents Components Diagrams and how they can be used to specify implementation artefacts: Source Code, Executable Releases and Physical Databases. Implementation modelling is covered in the slides 527 through 540.

#### 5.4.5. Deployment

This section teaches how Component Diagrams and Nodes can be used to describe the deployment environment of a system. Slides 541 to 554 contain this Section.

#### 5.5. Unified Process

The Unified Process (UP) is taught in this section as a process supporting Object Oriented Analysis and Design with UML. The section explains the details of the Unified Process: Core Workflows and Life Cycle Phases. Finally, it shows which UML models are relevant to specific Workflows and Phases of the UP. Slides 555 through 579 contain this Section.

#### 5.6. Tools

This section presents some of the tools for UML modelling: Enterprise Architect, Magic Draw, Poseidon and Rational Rose. The basic features of each tool are presented, as is the comparative analysis of the tools. This Section is presented in the slides 580 through 591.

#### 5.7. Summary

This section provides a summary of the entire course highlighting the major points in each of the sections. The summary section is given from the slide 592 to 618.

### 6. Assessment

Assessment comprises 15 multiple-choice questions provided at the end of the course to test the students' understanding of the various topics taught. Two sets of assessment questions are given in Appendices B.1 and B.2, with only slight differences between them. The assessment compliments the 55 tasks or assignments provided in the various sections.

### 7. Organization

The course consists of lectures, demonstrations and assignments:

- *lectures* – The lectures aim to incrementally build a body of concepts to establish the foundation for proper understanding and use of technology.
- *demonstrations* - Demonstrations illustrate the concepts introduced during the lectures with running code and small-size examples.



- *assignments* - During assignments, students are asked to perform a range of tasks with increasing level of difficulty and independence. They are encouraged to reuse the demonstrated code and examples in their assignments.

The course is designed to be taught for about 42 hours. In the context of the e-Macao Core Team Training, the course was taught over seven days, as follows: (1) Introduction and Object Orientation and UML Basics (2) UML Basics and Requirements, (3) Requirements, (4) Architecture, (5) Design, (6) Implementation and Deployment, and (7) Unified Process, Tools and Summary. For e-Macao Extended Team Training, a shorter version of the same course was taught over four days: (1) Introduction, Object Orientation and UML Basics, (2) Requirements, (3) Architecture and Design, and (4) Design, Implementation, Deployment, Unified Process, UML Tools and Summary.

---

## References

- 1) OMG Unified Modeling Language Specification, Object Management Group.
- 2) UML Bible, Tom Pender, John Wiley and Sons, 2003.
- 3) Object-Oriented Analysis and Design using UML, Simon Bennet, Steve McRobb and Ray Farmer, McGraw-Hill, 2002.
- 4) Guide to Applying the UML, Sinan Si Alhir, Springer, 2002.
- 5) Object-Oriented Software Engineering, Bernd Bruegge, Allen H Dutoit, Prentice Hall, 2000.
- 6) The Unified Modeling Language User Guide, Grady Booch, James Rumbaugh, Ivar Jacobson, Addison Wesley, 1999.
- 7) OO Software Development Using UML, UNU-IIST Tech Report 229.

## Appendix

### A. Slides

#### A.1. Introduction

# Object-Oriented Analysis and Design with UML

Adegboyega Ojo and Elsa Estevez

UNU-IIST

e-Macao-18-1-2

## The Course

- **objectives** - what do we intend to achieve?
- **outline** - what content will be taught?
- **resources** - what teaching resources will be available?
- **organization** - duration, major activities, daily schedule

e-Macao-18-1-3

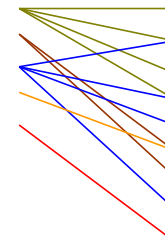
## Course Objectives

- 1) present the concepts of Object Orientation, as well as Object Oriented Analysis (OOA) and Design (OOD)
- 2) introduce the syntax, semantics and pragmatics of UML, and how to integrate it with the Unified Process
- 3) show how to articulate requirements using use cases
- 4) present the development of structural and behavioural diagrams for different views supported by UML
- 5) introduce patterns and frameworks and their applications in architecture and design tasks
- 6) present a comparative analysis of the major UML tools suitable for industrial-strength development

e-Macao-18-1-4

## Course Outline

- 1) Object Orientation
- 2) UML Basics
- 3) UML Modelling:
  - a) Requirements
  - b) Architecture
  - c) Design
  - d) Implementation
  - e) Deployment
- 4) Unified Process
- 5) UML Tools



### UML Diagrams:

1. use case
2. class
3. object
4. sequence
5. state
6. component
7. collaboration
8. activity
9. deployment

e-Macao-18-1-5

## Overview: Object Orientation

- 1) Background
- 2) Principles
- 3) Concepts
- 4) OO Analysis
- 5) OO Design

e-Macao-18-1-6

## Overview: UML Basics

- 1) UML outline
- 2) UML building blocks
  - a) elements
  - b) relationships
  - c) diagrams
- 3) modelling views

e-Macao-18-1-7

## Overview: Requirements

1. software requirements
2. use case modelling
  - a) use case diagrams
  - b) templates
3. conceptual modelling
  - a) class diagrams
  - b) object diagrams
4. behavioural modelling
  - a) sequence diagrams
  - b) statechart diagrams

e-Macao-18-1-8

## Overview: Architecture

- 1) software architecture
- 2) models
  - a) collaboration diagrams
  - b) component diagrams
- 3) frameworks and patterns

e-Macao-18-1-9

## Overview: Design

- 1) software design
- 2) design models
  - a) class diagrams
  - b) sequence diagrams
  - c) activity diagrams
  - d) statechart diagrams
- 3) design patterns

e-Macao-18-1-10

## Overview: Implementation

- 1) packages
- 2) components
- 3) component diagrams

e-Macao-18-1-11

## Overview: Deployment

- 1) nodes and components
- 2) deployment diagrams
- 3) use of deployment diagrams

e-Macao-18-1-12

## Overview: Unified Process

- 1) Unified Process
- 2) life cycle phases
- 3) workflows
- 4) workflows and UML diagrams

e-Macao-18-1-13

## Overview: UML Tools

- 1) CASE tools
- 2) Selected UML tools:
  - a) Magic Draw
  - b) Enterprise Architect
  - c) Poseidon
  - d) Rational Rose

e-Macao-18-1-14

## Course Resources

- 1) OMG Unified Modeling Language Specification, Object Management Group.
- 2) UML Bible, Tom Pender, John Wiley and Sons, 2003.
- 3) Object-Oriented Analysis and Design using UML, Simon Bennet, Steve McRobb and Ray Farmer, McGraw-Hill, 2002.
- 4) Guide to Applying the UML, Sinan Si Alhir, Springer, 2002.
- 5) Object-Oriented Software Engineering, Bernd Bruegge, Allen H Dutoit, Prentice Hall, 2000.
- 6) The Unified Modeling Language User Guide, Grady Booch, James Rumbaugh, Ivar Jacobson, Addison Wesley, 1999.
- 7) OO Software Development Using UML, UNU-IIST Tech Report 229.
- 8) Course materials

e-Macao-18-1-15

## Course Organization

Monday to Thursday:

09:00 – 10:20 lecture/lab

10:20 – 10:30 break

10:30 – 11:20 lecture/lab

11:20 – 11:30 break

11:30 – 13:00 lecture/lab

13:00 – 14:30 lunch time

14:30 – 15:30 lecture/lab

15:30 – 15:40 break

15:40 – 16:30 lecture/lab

16:30 – 16:40 break

16:40 – 17:45 lecture/lab (except Thursday)

e-Macao-18-1-16

## Course Assessment

Thursday, 16:45 – 17:45

15 multiple-choice questions

30 minutes allowed

maximum obtainable points – 30

grades	points
A	26-30
B	21-25
C	16-20
D	11-15
E	00-10

## A.2. Object Orientation

### A.2.1. Background

# Object Orientation

e-Macao-18-1-18

## Course Outline

### 1) Object Orientation

### 2) UML Basics

### 3) UML Modelling:

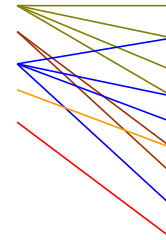
- a) Requirements
- b) Architecture
- c) Design
- d) Implementation
- e) Deployment

### 4) Unified Process

### 5) UML Tools

### UML Diagrams:

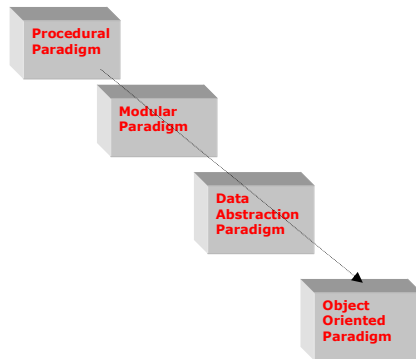
- 1. use case
- 2. class
- 3. object
- 4. sequence
- 5. state
- 6. component
- 7. collaboration
- 8. activity
- 9. deployment





e-Macao-18-1-19

## Towards Object Orientation



e-Macao-18-1-20

## What is Object Orientation?

### Definition

Object Orientation is about viewing and modelling the world/system as a set of interacting and interrelated *objects*.

Features of the OO approach:

- 1) the universe consists of interacting objects
- 2) describes and builds systems consisting of objects

## A.2.2. Principles

e-Macao-18-1-21

### OO Principles

- 1) abstraction
- 2) encapsulation
- 3) modularity
- 4) hierarchy

e-Macao-18-1-22

### Principle 1: Abstraction

A process allowing to focus on **most important** aspects while ignoring **less important** details.

**Abstraction** allows us to manage complexity by concentrating on essential aspects making an entity different from others.



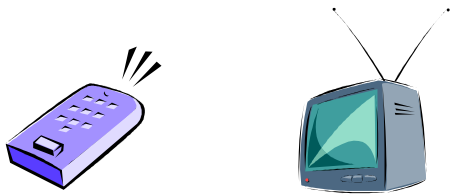
*An example of an order processing abstraction*

e-Macao-18-1-23

## Principle 2: Encapsulation

**Encapsulation** separates implementation from users/clients.

A client depends on the interface.

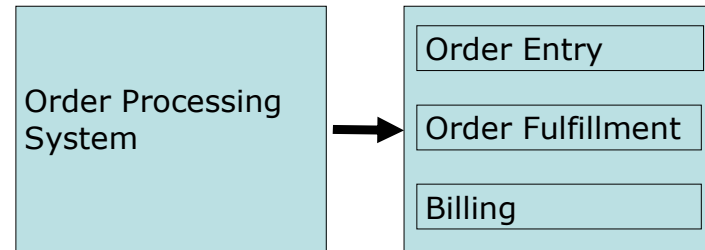


*Courtesy Rational Software*

e-Macao-18-1-24

## Principle 3: Modularity

**Modularity** breaks up complex systems into small, self-contained pieces that can be managed independently.

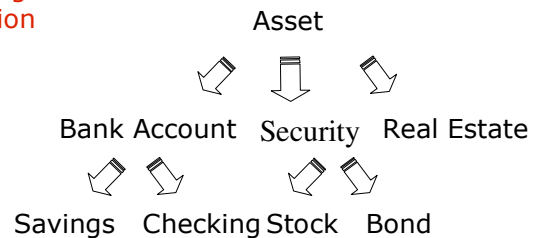


e-Macao-18-1-25

## Principle 4: Hierarchy

Ordering of abstractions into a tree-like structure.

Increasing  
abstraction



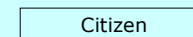
e-Macao-18-1-26

## Task 1

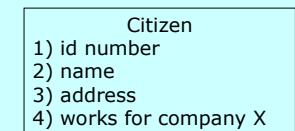
Which of the following models A and B is more abstract?

Justify your answer.

Model A:



Model B:



e-Macao-18-1-27

## Task 2

Provide an example of encapsulation:

- a) identify the interface
- b) identify the implementation

Tip:

interface: computer case

implementation: computer mother board

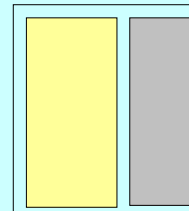
e-Macao-18-1-28

## Task 3

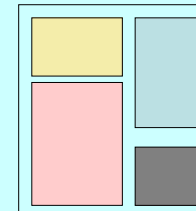
Suppose A and B are models of the same system.

Which of them is more modular?

Model A:



Model B:



e-Macao-18-1-29

## Task 4

Describe the structure of your agency as a hierarchy.

Include at least three levels.

### A.2.3. Concepts

e-Macao-18-1-30

## OO Concepts

- |                   |                   |
|-------------------|-------------------|
| 1. Object         | 10.Generalization |
| 2. Class          | 11.Super-Class    |
| 3. Attribute      | 12.Sub-Class      |
| 4. Operation      | 13.Abstract Class |
| 5. Interface      | 14.Concrete Class |
| 6. Implementation | 15.Discriminator  |
| 7. Association    | 16.Polymorphism   |
| 8. Aggregation    | 17.Realization    |
| 9. Composition    |                   |

e-Macao-18-1-31

## Concept 1: Objects

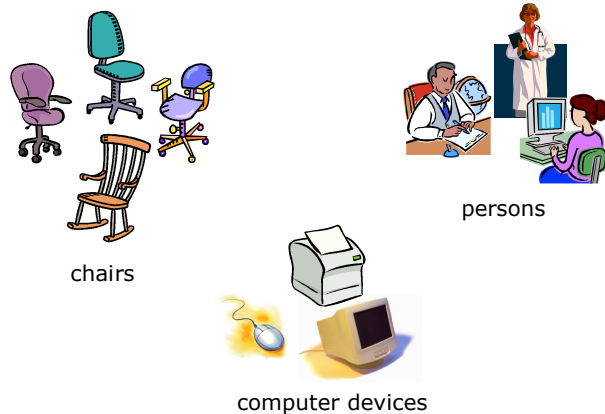
What is an **object**?

- 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

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

e-Macao-18-1-32

## Examples of Objects



e-Macao-18-1-33

## Object Definition

Two aspects:

- **Information:**
  - 1) has a unique identity
  - 2) has a description of its structure
  - 3) has a state representing its current condition
- **Behaviour:**
  - 1) what can an object do?
  - 2) what can be done to it?

e-Macao-18-1-34

## Object Definition Example



- 1) **information:**
  - a) serial number
  - b) model
  - c) speed
  - d) memory
  - e) status
- 2) **behaviour:**
  - a) print file
  - b) stop printing
  - c) empty the queue

e-Macao-18-1-35

## Concept 2: Class

What is a **class**?

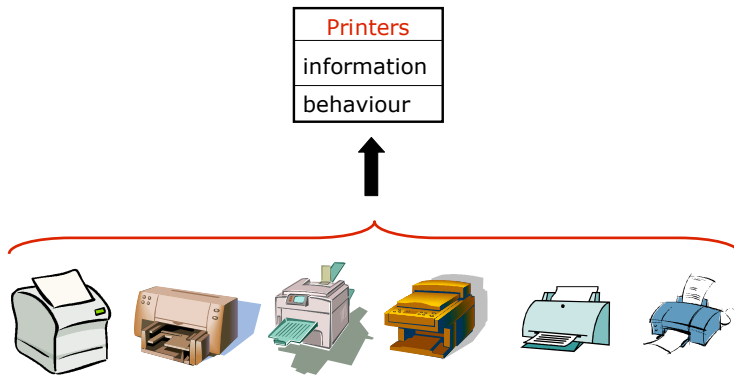
- 1) any uniquely identified abstraction of a set of logically related instances that share similar characteristics
- 2) rules that define objects
- 3) a definition or template that describes how to build an accurate representation of a specific type of objects

Examples: agency, citizen, car, etc.

Objects are created using class definitions as templates.

e-Macao-18-1-36

## Class Example



e-Macao-18-1-37

## Concept 3: Attribute

### Definition

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

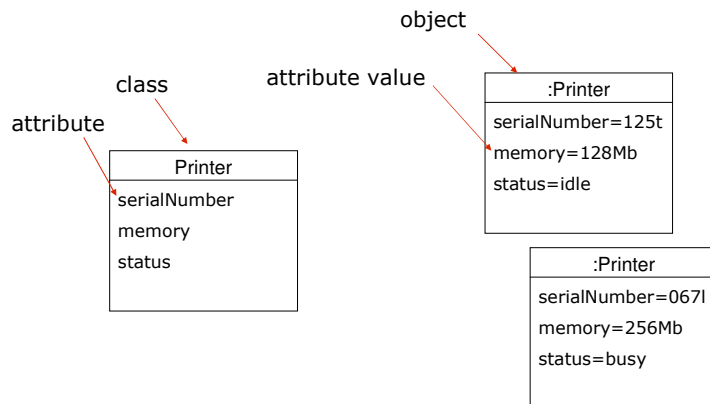
An attribute has a type and defines the type of its instances.

Only the object is able to change the values of its own attributes.

The set of attribute values defines the state of the object.

e-Macao-18-1-38

## Attribute Examples



e-Macao-18-1-39

## Concept 4: Operation

### Definition

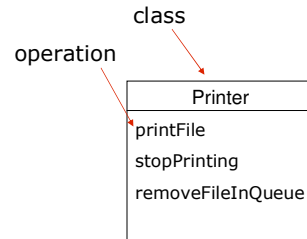
**Operation** is the implementation of a service that can be requested from any object of a given class.

An operation could be:

1. a question - does not change the values of the attributes
2. a command - may change the values of the attributes

e-Macao-18-1-40

## Operation Example



e-Macao-18-1-41

## Task 5

Which of the following statements are true?

For those that are false, explain why.

- 1) Trainee is an example of a class.
- 2) Dora Cheong is an example of a class.
- 3) createJob is an example of an object.
- 4) deleteFile is an example of an operation
- 5) name is an attribute

e-Macao-18-1-42

## Task 6

Suppose we wish to model an application for issuing business registration licenses.

Identify:

- 1) three classes for the model
- 2) at least three attributes for each class

e-Macao-18-1-43

## Applying Abstraction

Abstraction in Object Orientation:

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



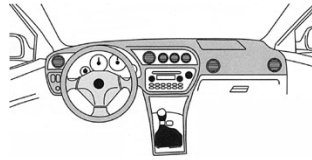
*Courtesy: XML Bible*



e-Macao-18-1-44

## Concept 5: 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



e-Macao-18-1-45

## Interface Example

```
package edu.unu.list;

import java.util.Date;
import java.util.Properties;

/**
 * @author elsa
 */
public interface BusinessCalendar {

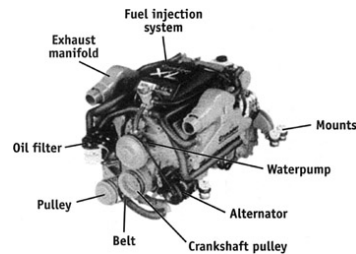
    /**
     * Get information on the properties of the business calendar
     * @return Properties of the business calendar
     */
    public Properties getBusinessCalendarProperties();

    /**
     * Add an appointment to the calendar
     * @param date the appointment date
     * @return a confirming date
     */
    public Date add(Date date);
}
```

e-Macao-18-1-46

## Concept 6: 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 upon which the behaviour depends



e-Macao-18-1-47

## Implementation Example

```
public class BusinessCalendar {

    Day[] weekdays = null;
    List holidays = null;

    private static Properties businessCalendarProperties = null;
    public static Properties getBusinessCalendarProperties() {
        if (businessCalendarProperties == null) {
            businessCalendarProperties = new Properties();
            InputStream is = ClassLoaderUtil.getStream("jbpm.business.calendar.properties", "org/jbpm/calendar");
            try {
                businessCalendarProperties.load(is);
            } catch (IOException e) {
                throw new RuntimeException("couldn't load business.calendar.properties", e);
            }
        }
        return businessCalendarProperties;
    }

    public Date add(Date date, Duration duration) {
        Date end = null;
        if (duration.isBusinessTime()) {
            DayPart dayPart = findDayPart(date);
            boolean isInBusinessHours = (dayPart != null);
            if (!isInBusinessHours) {
                Object[] result = new Object[2];
                findDay(date).findNextDayPartStart(0, date, result);
                date = (Date) result[0];
                dayPart = (DayPart) result[1];
            }
            end = dayPart.add(date, duration);
        } else {
            end = new Date(date.getTime() + duration.milliseconds);
        }
        return end;
    }
}
```

e-Macao-18-1-48

## Relations and Links

### Relationships:

- between classes (relations)
- between objects (links)

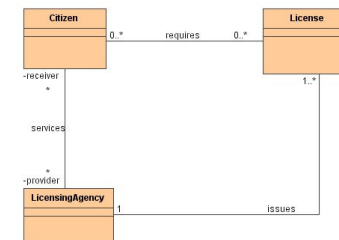
Three kinds of relations between classes:

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

e-Macao-18-1-50

## Concept 7: Association

1. the simplest form of relation between classes
2. peer-to-peer relations
3. one object is aware of the existence of another object
4. implemented in objects as references



e-Macao-18-1-51

## Association Examples

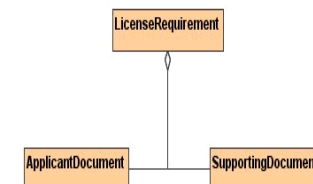
Associations between classes A and B:

- 1) A is a physical or 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

e-Macao-18-1-52

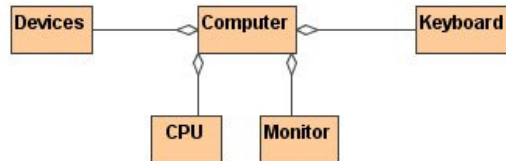
## Concept 8: Aggregation

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



e-Macao-18-1-53

## Aggregation Example



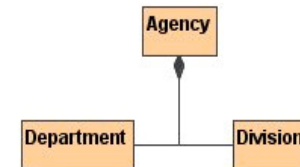
A CPU is part of a computer.

CPU, devices, monitor and keyboard are assembled to create a computer.

e-Macao-18-1-54

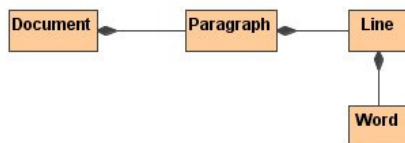
## Concept 9: Composition

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



e-Macao-18-1-55

## Composition Example



A word cannot exist if it is not part of a line.

If a paragraph is removed, all lines of the paragraph are removed, and all words belonging to that lines are removed.

e-Macao-18-1-56

## Task 7

Identify which of the following statements are true.

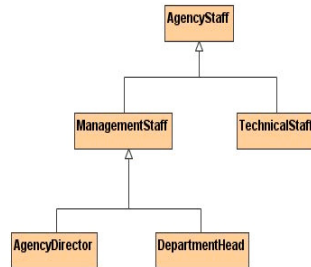
For those that are false, explain why.

- 1) there is an association between Trainee and Course
- 2) there is a composition between Course and Professor
- 3) there is an aggregation between Course and Venue

e-Macao-18-1-58

## Concept 10: 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) generalization enables inheritance
- 4) specialization is the opposite of generalization
- 5) not an association



e-Macao-18-1-60

## Concept 11: Super-Class

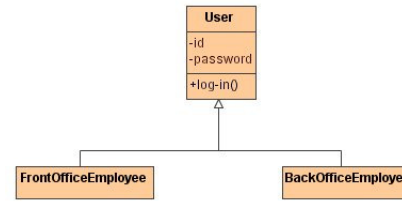
### Definition

**Super-Class** is a class that contains the features common to two or more classes.

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

e-Macao-18-1-59

## Generalization Example



Common features are defined in User.

FrontOfficeEmployee and BackOfficeEmployee inherit them.

e-Macao-18-1-61

## Concept 12: Sub-Class

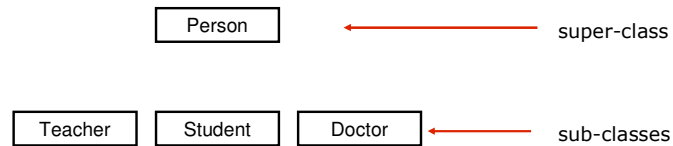
### Definition

**Sub-Class** is a class that contains at least the features of its super-class(es).

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

e-Macao-18-1-62

## Super/Sub-Class Example



e-Macao-18-1-63

## Concept 13: Abstract Class

1. a class that lacks a complete implementation - provides operations without implementing some methods
2. cannot be used to create objects; cannot be instantiated
3. a concrete sub-class must provide methods for unimplemented operations

e-Macao-18-1-64

## Concept 14: Concrete Class

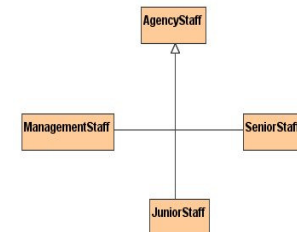
1. has methods for all operations
2. can be instantiated
3. methods may be:
  - a) defined in the class or
  - b) inherited from a super-class

e-Macao-18-1-65

## Concept 15: Discriminator

**Discriminator** – an attribute that defines sub-classes

Example: "status" of agency staff is a possible discriminator to derive "management", "senior" and "junior" sub-classes.

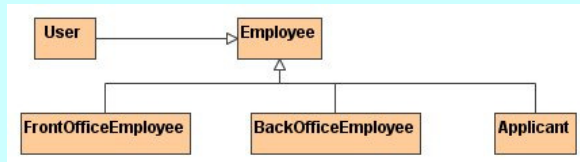


e-Macao-18-1-66

## Task 9

Suppose we wish to model an e-service application for a government agency. Does this diagram reasonably model the relationship between the entities User, Employee, Front-Office Employee, Back-Office Employee and Applicant?

If not, provide a more appropriate model.



e-Macao-18-1-67

## Task 10

- Add a generalization for the classes defined in Task 6.
- Identify the super-class and one sub-class.
- Do you have any abstract class? Justify.
- Which discriminator is used in your generalization hierarchy?

e-Macao-18-1-68

## Concept 16: Polymorphism

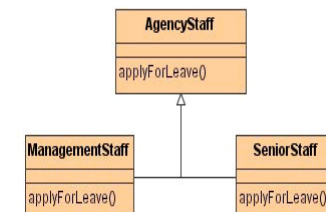
1. ability to dynamically choose the method for an operation at run-time or service-time
2. facilitated by encapsulation and generalization:
  - a) **encapsulation** – separation of interface from implementation
  - a) **generalization** – organizing information such that the shared features reside in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

e-Macao-18-1-69

## Polymorphism Example

Many ways of doing the same thing!

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



e-Macao-18-1-70

## Task 11

Consider the generalization hierarchy provided in Task 10.

Introduce an operation for the super-class which could be implemented in different ways by its sub-classes.

Provide an example of polymorphism. Explain.

e-Macao-18-1-72

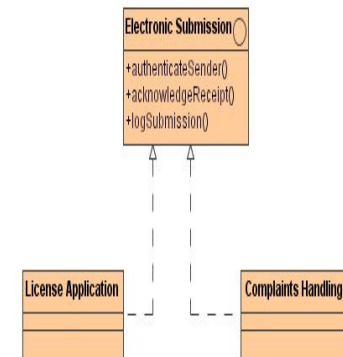
## Task 12

Using realization model the fact that vehicles and aircrafts must be to accelerate, decelerate, brake and park.

e-Macao-18-1-71

## Concept 17: Realization

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



**A.2.4. Object Oriented Analysis**

e-Macao-18-1-73

## OO Analysis and Design

A methodology for system/software modelling and design using Object-Oriented concepts.

Consists of two parts:

- 1) Object Oriented Analysis
- 2) Object Oriented Design

e-Macao-18-1-74

## Object Oriented Analysis

- 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 problem domain classes
- 5) concerned with developing an object-oriented model of the problem domain
- 6) identified objects reflect the entities that are associated with the problem to be solved



e-Macao-18-1-75

## OOA Definition

### Definition

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

[Software Engineering Institute]

e-Macao-18-1-76

## 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 artifacts that are independent of the analysis method or programming language
- 3) **productivity**: direct mapping to the features implemented in Object Oriented Programming Languages

### A.2.5. Object Oriented Design

e-Macao-18-1-77

## Object Oriented Design

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

e-Macao-18-1-78

## OOD Design

Definition [SEI]

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

The same set of benefits as those in OOA.

e-Macao-18-1-79

## OOD Process

Stages in OOD:

- 1) 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

Notes on the 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

e-Macao-18-1-81

## Summary 1

Object is any abstraction that models a single thing in a universe with some defined 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 three types of relations:

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

e-Macao-18-1-80

## Task 13

List two basic differences between traditional systems analysis and object-oriented analysis.

e-Macao-18-1-82

## Summary 2

Object Orientation is characterized by:

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

e-Macao-18-1-83

## Summary 3

**Object Oriented Analysis** is concerned with specifying system requirements and analysing the application domain.

**Object Oriented Design** is concerned with implementing the requirements identified during OOA in the application domain.

## A.3. UML Basics

### A.3.1. UML Background

# UML Basics

e-Macao-18-1-85

## Course Outline

1) Object Orientation

2) UML Basics

3) UML Modelling:

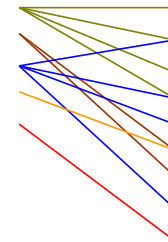
- a) Requirements
- b) Architecture
- c) Design
- d) Implementation
- e) Deployment

4) Unified Process

5) UML Tools

UML Diagrams:

- 1. use case
- 2. class
- 3. object
- 4. sequence
- 5. state
- 6. component
- 7. collaboration
- 8. activity
- 9. deployment



e-Macao-18-1-86

## What Is Modelling?

Modelling involves:

- 1) representation or simplification of reality
- 2) providing a blueprint of a system

e-Macao-18-1-87

## Why Model?

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

e-Macao-18-1-88

## Modelling Principles

- 1) the choice of models affects how a problem is tackled
- 2) every model may be expressed at different levels of abstraction
- 3) effective models are connected to reality
- 4) no single model is sufficient to describe non-trivial systems

e-Macao-18-1-89

## What is UML?

UML = Unified Modelling Language

A language for visualizing, specifying, constructing and documenting artifacts of software-intensive systems.

Examples of artifacts: requirements, architecture, design, source code, test cases, prototypes, etc.

UML is suitable for modelling various kinds of systems:

1. enterprise information systems,
2. distributed web-based,
3. real-time embedded system, etc.

e-Macao-18-1-90

## UML – Specification Language

Provides views for development and deployment.

UML is process-independent.

UML is recommended for use with the processes that are:

- 1) use-case driven
- 2) architecture-centric
- 3) iterative
- 4) incremental

e-Macao-18-1-90

## UML – Specification Language

Provides views for development and deployment.

UML is process-independent.

UML is recommended for use with the processes that are:

- 1) use-case driven
- 2) architecture-centric
- 3) iterative
- 4) incremental

e-Macao-18-1-91

## Goals of UML

- 1) provide modelers with an expressive, 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 basis for understanding specification languages
- 5) encourage the growth of the object tools market
- 6) supports higher level of development with concepts such as components frameworks or patterns

e-Macao-18-1-92

## History of UML

- 1) started as a unification of the Booch and Rumbaugh methods - *Unified Method v. 0.8* (1995)
- 2) Jacobson contributes to produce UML 0.9, 1996
- 3) UML Partners work with three Amigos to propose UML as a standard modelling language to OMG in 1996.
- 4) UML partners tender their proposal (UML 1.0) to OMG in 1997, and 9 months later submit final UML 1.1.
- 5) Minor revision is UML 1.4 adopted in May 2001, and the most recent revision is UML 1.5, March 2003.
- 6) UML 2.0 released by the end 2004.

### A.3.2. UML Building Blocks

e-Macao-18-1-93

## 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

They will be used to represent complex structures.

e-Macao-18-1-94

## Elements

Four basic types of elements:

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

They will be used to specify well-formed models.



e-Macao-18-1-95

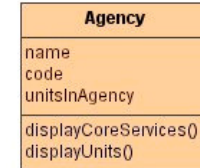
## Structural Elements

- a) static part of the model to represent conceptual elements
- b) “nouns” of the model
- c) seven kinds of structural elements:
  - 1) class
  - 2) interface
  - 3) collaboration
  - 4) use case
  - 5) active class
  - 6) component
  - 7) node

e-Macao-18-1-96

## Element 1: Class

- 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



e-Macao-18-1-97

## Element 2: Interface

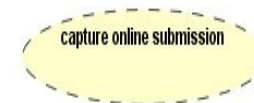
- 1) collection of operations that specifies a service of a class
- 2) describes the externally visible behaviour, partial or complete, of a class
- 3) defines a set of operation signatures but not their implementations
- 4) rendered as a circle with a name



e-Macao-18-1-98

## Element 3: 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 of cooperation that make up a system
- 5) represented as a named ellipse drawn with a dashed line



e-Macao-18-1-99

## Element 4: 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



e-Macao-18-1-100

## Element 5: Active Class

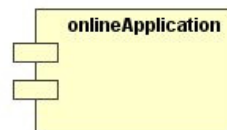
- 1) a class whose objects own one or more processes or threads and therefore can initiate an action
- 2) a class whose objects have concurrent behaviour with other objects
- 3) it also can be used to represent processes and threads
- 4) graphically, an active class is rendered just like a class drawn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



e-Macao-18-1-101

## Element 6: 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 can also be used to represent applications, files, libraries, pages and tables

e-Macao-18-1-102

## Element 7: Node

- 1) a physical element, 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



e-Macao-18-1-103

## Behavioural Elements

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

e-Macao-18-1-104

## Element 8: Interaction

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

saveapplication() →

e-Macao-18-1-105

## Element 9: State Machine

- 1) a sequence of states an object or 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: states, transition, events and activities
- 4) presented as a rounded rectangle with a name and sub-states



Interactions and state machines are associated with structural elements such as classes, collaborations and objects.

e-Macao-18-1-106

## Element 10: Package

Grouping element of UML:

- organizes diagrams
- primary kind of grouping and decomposition
- conceptual, only available at development time
- graphically represented as a tabbed folder

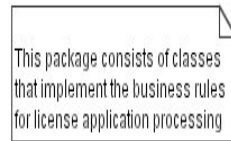


e-Macao-18-1-107

## Element 11: Note

Annotation element:

- comments added to models for better explanation or illumination on specific elements
- used primarily for annotation e.g. rendering constraints and comments attached to elements or collections of elements
- presented as a rectangle with a dog-eared corner
- may include both textual and graphical comments



e-Macao-18-1-108

## Task 14

Identify the UML element which best models each of the following entities:

- 1) Jackie Chan
- 2) a set of operations for validating dates
- 3) a module capable of validating dates based on the operations in 2
- 4) a set of trainees
- 5) "withdraw money" request from an ATM system
- 6) a computer server that hosts the payroll system

e-Macao-18-1-109

## Relationships

Four basic types of relationships:

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

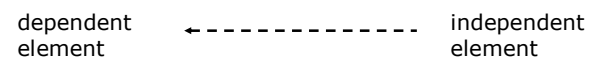
Meanings are consistent with the basic Object Oriented relationship types described earlier.

e-Macao-18-1-110

## Relationship 1: Dependency

A semantic relationship between two elements in which a change to one element (independent element) may affect the meaning of another (dependent element).

Given as a directed dashed line possibly with a label:



e-Macao-18-1-111

## Relationship 2: Association

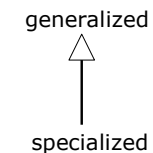
- 1) a structural relationship describing a set of links
- 2) links are connections between objects
- 3) aggregation is a special type of association depicting the whole-part relationship
- 4) association is presented as a solid line, possibly directed, labelled and with adornments (multiplicity and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



e-Macao-18-1-112

## Relationship 3: Generalization

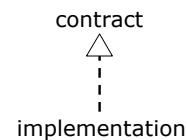
- 1) a relationship in which objects of a specialized element (child) are substitutable for objects of a generalized element (parent)
- 2) child elements share the structure/behaviour of the parent
- 3) rendered graphically as a solid line with hollow arrowhead pointing to the parent



e-Macao-18-1-113

## Relationship 4: 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/components
  - b) uses cases versus realizing collaborations
- 3) graphically depicted as a dashed arrow with hollow head, a cross between dependency and generalization



e-Macao-18-1-114

## Variants of Relationships

Variations of these four relationship types include:

- 1) refinement
- 2) trace
- 3) include
- 4) extend

e-Macao-18-1-115

## Task 15

Define the relationship type which best relates the following classes:

- 1) Employee – Administrative officers
- 2) Employee – Office
- 3) Application Software – Operating system
- 4) PC - Device

e-Macao-18-1-116

## Diagrams

A diagram is a graph presentation of a set of elements and relationships where:

- a) nodes are elements
- b) edges are relationships

Can visualize a system from various perspective, thus is a projection of a system.

e-Macao-18-1-117

## Diagram Types

UML is characterized by nine major diagrams:

- 1) class
- 2) object
- 3) use case
- 4) sequence
- 5) collaboration
- 6) statechart
- 7) activity
- 8) component
- 9) deployment

e-Macao-18-1-118

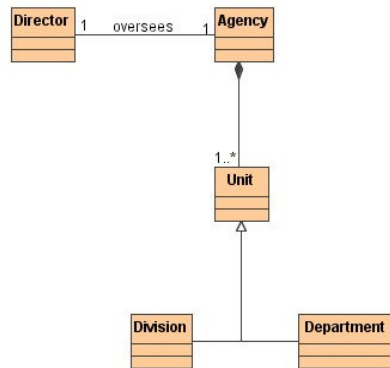
## Diagram 1: Class

**Class Diagrams:**

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

e-Macao-18-1-119

## Class Diagram Example



e-Macao-18-1-120

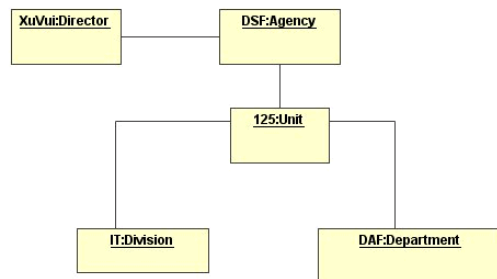
## Diagram 2: Object

### Object Diagrams:

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

e-Macao-18-1-121

## Object Diagram Example



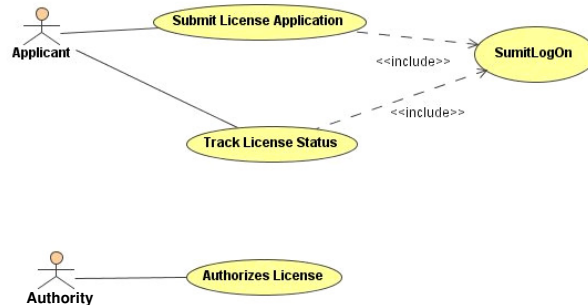
e-Macao-18-1-122

## Diagram 3: Use Case

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

e-Macao-18-1-123

## Use Case Diagram Example



e-Macao-18-1-124

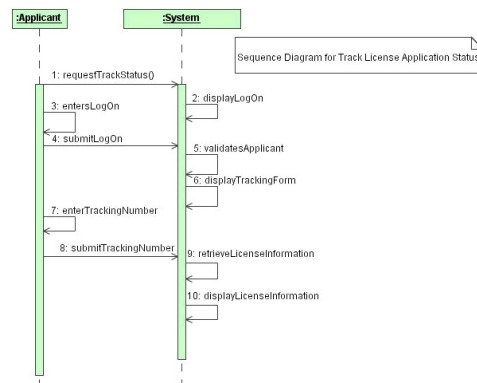
## Diagram 4: Sequence

### Sequence Diagrams:

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

e-Macao-18-1-125

## Sequence Diagram Example



e-Macao-18-1-126

## Diagram 5: Collaboration

### Collaboration Diagram:

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

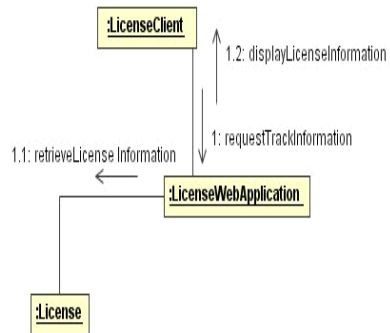
### Sequence and collaboration diagrams:

- are jointly called interaction diagrams
- can be transformed one into another



e-Macao-18-1-127

## Collaboration Diagram Example



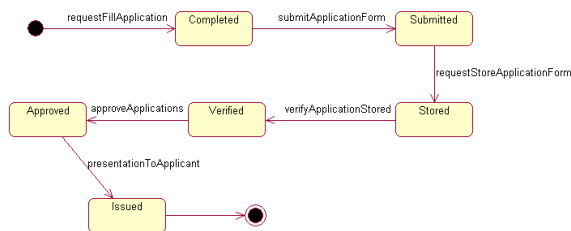
e-Macao-18-1-128

## Diagram 6: Statechart

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

e-Macao-18-1-129

## Statechart Diagram Example



e-Macao-18-1-130

## Diagram 7: Activity

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

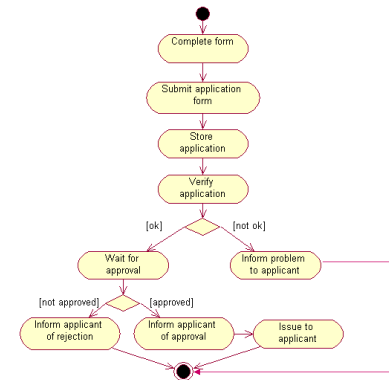
e-Macao-18-1-130

## Diagram 7: Activity

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

e-Macao-18-1-131

## Activity Diagram Example



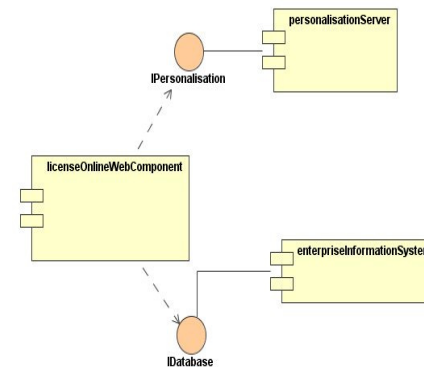
e-Macao-18-1-132

## Diagram 8: Component

- 1) shows the organization and dependencies amongst a set of components
- 2) addresses static implementation view of a system
- 3) shows how components map to one or more classes, interfaces and collaborations

e-Macao-18-1-133

## Component Diagram Example



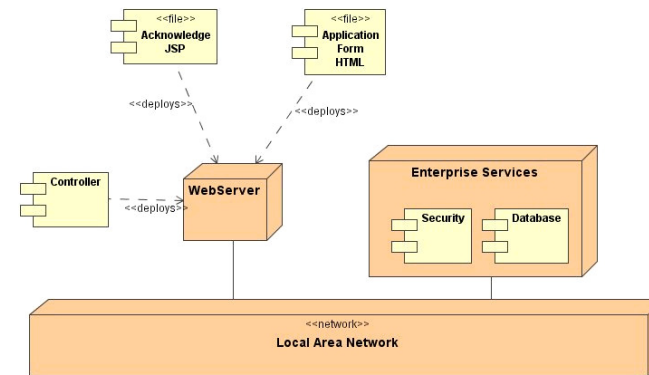
e-Macao-18-1-134

## Diagram 9: Deployment

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

e-Macao-18-1-135

## Deployment Diagram Example



e-Macao-18-1-136

## Task 16

Consider a software process consisting of the following activities: requirement gathering, object oriented analysis, object design, implementation and deployment.

List the diagrams that are essential for each of these activities.

Provide justifications for your choice of diagrams.

e-Macao-18-1-137

## Task 17

Select the best answer:

The class diagram:

- a) is the first model created in the project
- b) is created after the other models
- c) is used to specify objects and generate code
- d) is used to create the sequence and the collaboration diagrams.

e-Macao-18-1-138

## Task 18

---

Select the best answer:

The sequence diagram models:

- a) the sequence of activities to implement the model
- b) the way that objects communicate
- c) the relationships among objects
- d) the order in which the class diagram is constructed

e-Macao-18-1-139

## Task 19

---

Select the best answer:

The collaboration diagram:

- a) is a unique view of object behaviour
- b) models the connections between different views
- c) models the relationships between software and hardware components
- d) models the way objects communicate

### A.3.3. Modelling Views

e-Macao-18-1-140

## Modelling Views

Modelling views – different perspectives on the system:

- Use Case View
- Design View
- Process View
- Implementation View
- Deployment View

A view is essentially a set of diagrams.

e-Macao-18-1-141

## View 1: Use Case

**Use Case View** describes the behaviour of the system as seen by its end users, analysts and testers.

This view shapes the system architecture.

e-Macao-18-1-142

## View 2: Design

**Design View** encompasses:

- 1) classes,
- 2) interfaces and
- 3) collaborations

that form the vocabulary of the problem and its solution.

e-Macao-18-1-143

## View 3: Process

**Process View** encompasses threads and processes that form the system's concurrency and synchronization mechanisms.

This view addresses:

- 1) performance,
- 2) scalability and
- 3) throughput

of the system.

e-Macao-18-1-144

## View 4: Implementation

**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.

e-Macao-18-1-145

## View 5: Deployment

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

e-Macao-18-1-146

## Selecting Views

Different modelling views for different system types:

- 1) monolithic systems
- 2) distributed systems
- 3) etc.

e-Macao-18-1-147

## Monolithic Systems

Two views are relevant:

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

e-Macao-18-1-148

## Distributed Systems

All five views are relevant:

- 1) **Use case View**:
  - a) use case diagrams
  - b) activity diagrams
- 2) **Design View**:
  - a) class diagrams
  - b) interaction diagrams
  - c) statechart diagram
- 3) **Process View**:
  - a) class diagram
  - b) interaction diagrams
- 4) **Implementation View**: component diagrams
- 5) **Deployment View**: deployment diagrams

e-Macao-18-1-149

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

e-Macao-18-1-150

## Summary 2

---

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

Categories of elements in UML include:

1. structural
2. behavioural
3. grouping
4. annotation

There are four basic types of relationships in UML:

1. dependency
2. association
3. generalization
4. realization

e-Macao-18-1-151

## Summary 3

---

UML provides 9 diagrams for modelling:

1. class diagrams
2. object diagrams
3. use case diagrams
4. sequence diagrams
5. collaboration diagrams
6. statechart diagrams
7. activity diagrams
8. component diagrams
9. deployment diagrams

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



## A.4. Requirements

### A.4.1. Software Requirements

# Requirements Modelling

e-Macao-18-1-153

## Course Outline

1) Object Orientation

2) UML Basics

3) UML Modelling:

a) Requirements

b) Architecture

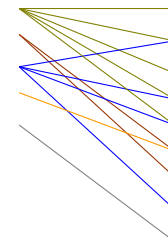
c) Design

d) Implementation

e) Deployment

4) Unified Process

5) UML Tools



UML Diagrams:

1. use case
2. class
3. object
4. sequence
5. state
6. component
7. collaboration
8. activity
9. deployment

e-Macao-18-1-154

## Requirements

### Definition

A **requirement** is:

- 1) a function that a system must perform
- 2) a desired characteristic of a system
- 3) a statement about the proposed system that all stakeholders agree that must be true in order for the customer's problem to be adequately solved.

e-Macao-18-1-155

## Requirements Process

Typically includes:

- 1) **elicitation** of requirements
- 2) **modelling** and **analysis** of requirements
- 3) **specification** of requirements
- 4) **validation** of requirements
- 5) requirements **management**

The process is not linear!

e-Macao-18-1-156

## Functional or Non-Functional?

### Functional requirements:

- 1) describe interaction between a system and its environment
- 2) describe how a system should behave under certain stimuli

### Non-functional requirements:

- 1) describe restrictions on a system that limit the choices for its construction as a solution to a given problem

e-Macao-18-1-157

## Requirements Types

- 1) functional
- 2) interface
- 3) data
- 4) human engineering
- 5) qualification
- 6) operational
- 7) design constraints
- 8) safety
- 9) security, etc.

e-Macao-18-1-158

## Requirements Specification

Requirements must be expressed precisely.

Requirements specification should be:

- 1) correct
- 2) consistent
- 3) feasible
- 4) verifiable
- 5) complete
- 6) traceable

e-Macao-18-1-159

## Requirements Standards

Various standards are available:

- 1) IEEE P1233/D3 Guide
- 2) IEEE Std. 1233 Guide
- 3) IEEE Std. 830-1998
- 4) ISO/IEC 12119-1994
- 5) IEEE std 1362-1998 (ConOps)

e-Macao-18-1-159

## Requirements Standards

Various standards are available:

- 1) IEEE P1233/D3 Guide
- 2) IEEE Std. 1233 Guide
- 3) IEEE Std. 830-1998
- 4) ISO/IEC 12119-1994
- 5) IEEE std 1362-1998 (ConOps)

e-Macao-18-1-160

## Requirements Template

req. id	F44
category	Functional
description	The applicants of social benefits, processing staff and approving authority shall be able to register in the system. Registration shall involve creating an account for the user for the purpose of authentication and personalisation of the system services. The agency staff users will be created by the system administrator. Applicants can register themselves by an option provided on the website.
terms	Account, Applicant, Authentication, Registration, User
justification	Registration is essential for subsequent authentication and personalisation functions.
priority	High
dependencies	F43
documents	
feasibility argument	There are mature solutions (e.g. directory services) for implementing the requirement.
verification method	When the system registers a new applicant who does not have an account, an account for the user is created by the system.

e-Macao-18-1-161

## Task 20

- 1) Enumerate three functional requirements for the application issuing business registration licenses.
- 2) Enumerate two non-functional requirements for the same application.
- 3) Using the template provided earlier specify one of the functional requirements described in the first point.

e-Macao-18-1-162

## Glossary 1

- 1) a set of terms that are defined and understood to form the basis for communication
- 2) a dictionary to carry out modelling
- 3) its purpose is to clarify the meaning of terms or to have the shared understanding of the terms amongst team members
- 4) is created during requirements definition, use case identification and conceptual modelling
- 5) is maintained throughout development

e-Macao-18-1-163

## Glossary 2

It is usually the central place for:

- 1) definitions of key concepts
- 2) clarification of ambiguous terms and concepts
- 3) explanations of jargons
- 4) description of business events
- 5) description of software actions

There is no specific format for glossaries:

- 1) reference identification for terms
- 2) definitions
- 3) categories
- 4) cross references

e-Macao-18-1-164

## Glossary Example

Id	Term	Definition
1	Agency	A government organization providing services to citizens, businesses and other government agencies.
2	Authentication	Proving a user's identity. To be able to access a Website or resource, a user must provide authentication via a password or some combination of tokens, biometrics and passwords.
3	Authorization	The act of granting approval. Authorization to resources or information within an application can be based on simple or complex access control methods.
4	Booking	A process of arranging a date and time when a citizen can visit a social welfare centre.
5	Bulletin	A message created by an agency that contains information about the social benefit service they want to post for advertisement.
6	Bulletin Board	A place on a computer system where citizens can read messages. All agencies can add their own messages.

e-Macao-18-1-165

## Task 21

---

- Define a glossary containing two definitions for the business license issuing application.

### A.4.2. Use Case Modelling

## Requirements Modelling

### Use Case Diagrams

e-Macao-18-1-167

## Course Outline

1) Object Orientation

2) UML Basics

3) UML Modelling:a) Requirements

b) Architecture

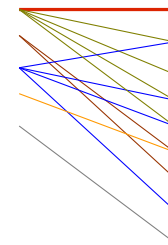
c) Design

d) Implementation

e) Deployment

4) Unified Process

5) UML Tools



UML Diagrams:

1. use case

2. class

3. object

4. sequence

5. state

6. component

7. collaboration

8. activity

9. deployment

e-Macao-18-1-168

## Use Cases

- 1) describe or capture **functional requirements**
- 2) represent the desired behaviour of the system
- 3) identify users (actors) of the system and associated processes
- 4) are the basic building blocks of use case diagrams
- 5) tie requirements phase to other development phases

e-Macao-18-1-169

## Use Case Definition 1

A use case:

1. is a collection of task-related activities describing a discrete chunk of a system
2. describes a set of actions sequences that a system performs to present an observable result to an actor
3. describes a system from an external usage viewpoint

e-Macao-18-1-170

## Use Case Definition 2

Key attributes of a use case:

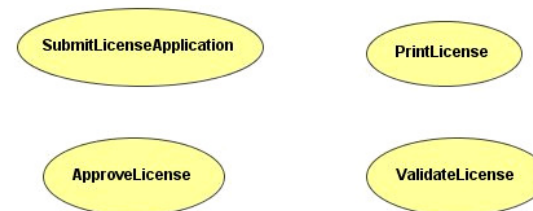
- 1) description
- 2) action sequence
- 3) includes variants
- 4) produces observable results

A use case does not describe:

- 1) user interfaces
- 2) performance goals
- 3) non-functional requirements

e-Macao-18-1-171

## Use Case Examples



e-Macao-18-1-172

## Use Case Relationships

Use cases are organized by relationships.

Three kinds:

1. generalization
2. include
3. extend

e-Macao-18-1-173

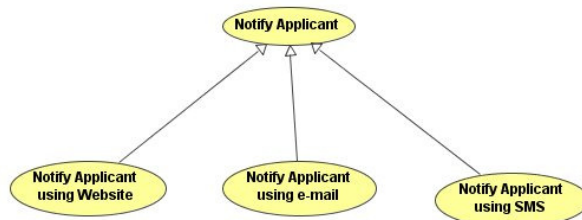
## Relationship 1: Generalization

Generalization of use cases:

- the same meaning as before
- a more specialized use case is related to a more general use case

e-Macao-18-1-174

## Generalization Example



e-Macao-18-1-175

## Relationship 2: Include

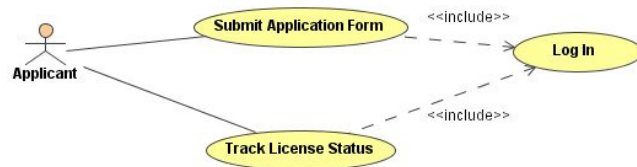
Include relationship between use cases:

- the base use case explicitly incorporates the behaviour of another use case at a location specified in the base
- the include relationship never stands alone, but is instantiated as part of some larger base of use cases
- rendered using the "include" stereotype



e-Macao-18-1-176

## Include Example



e-Macao-18-1-177

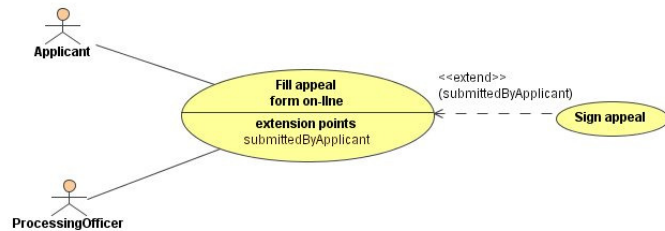
## Relationship 3: Extend

Extend relationship between use cases:

- the base use case implicitly incorporates the behaviour of another use case at a location specified by the extending use case (**extension point**)
- the base use case may stand alone and usually executes without regards to extension points
- depending on the system behaviour, the extension use case will be executed or not
- rendered using the "**extend**" stereotype

e-Macao-18-1-178

## Extend Example



e-Macao-18-1-179

## Actor

### Definition

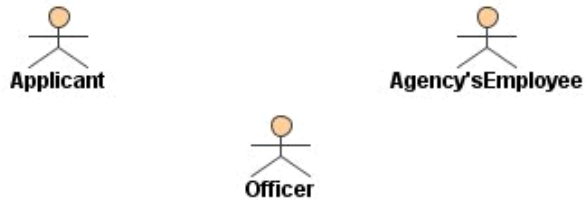
**Actor** is anyone or anything that interacts with the system causing it to respond to events.

An actor:

- is something or somebody that stimulates the system to react or respond to its request
- is something we do not have control over
- represents a coherent set of roles that the external entities to the system can play
- represents any type of a system's user

e-Macao-18-1-180

## Actor Examples



e-Macao-18-1-181

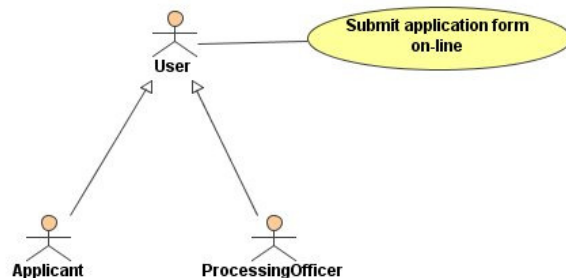
## Notes about Actors

- 1) actors **stimulate** a system providing input events and/or **receive** something output from the system
- 2) actors **communicate** with a system by sending and receiving messages to/from it, while performing use cases
- 3) actors model anything that needs to interact with the system to exchange information – human users, computer systems and others
- 4) a user may act as one or several actors as it interacts with the system, while several individual users may act as different instances of one and the same actor

e-Macao-18-1-182

## Relationship between Actors

It is also possible to relate actors using generalizations.



e-Macao-18-1-183

## Types of Actors

Initiator versus participant:

- When there is more than one actor in a use case, the one that generates the stimulus is called the **initiator** and the others are **participants**.

Primary versus secondary:

- The actor that directly interacts with the system is called the **primary** actor, others are called **secondary** actors.

e-Macao-18-1-184

## Task 22

- Based on the application for issuing business registration licenses define three use cases using a UML tool.
- Define a use case that extends one of the former use cases.
- Define a use case that can be included in one of the former use cases.

e-Macao-18-1-185

## Task 23

Select the best answer:

An actor is:

- a) a person
- b) a job title
- c) a role
- d) a system

e-Macao-18-1-186

## Task 24

- Add actors to the diagram produced in the previous Task.
- For each of the use cases in the previous task, enumerate all actors involved and indicate who is the initiator.

e-Macao-18-1-187

## Use Case Diagrams

- 1) a diagram that shows a set of use cases and actors, and their relationships
- 2) is central to modelling the functions of the system
- 3) is used to visualize the functions of a system, so that:
  - a) users can comprehend how to use the system and
  - b) developers can understand how to implement it
- 4) puts everything together

e-Macao-18-1-188

## Use Case Diagram Parts

A use case diagram commonly contains:

- 1) use cases
- 2) actors
- 3) relationships:
  - a) dependency - between use cases
  - b) generalization - between use cases or actors
  - c) association - between use cases and actors

e-Macao-18-1-189

## Identifying Use Cases

Use cases describe:

1. the functions that the user wants a system to accomplish
2. the operations that create, read, update, delete information
3. how actors are notified of the changes to the internal state and how they notify the system about external events

e-Macao-18-1-190

## Identifying Actors

To determine who are the actors,  
we try to answer the following questions:

- 1) who uses the system?
- 2) who gets information from the system?
- 3) who provides information to the system?
- 4) who installs, starts up or maintains the system?

e-Macao-18-1-191

## Naming Use Cases

Use concrete verb-noun phrases:

- 1) a weak verb may indicate uncertainty, a strong verb may clearly identify the action taken:
  - a) **strong verbs**: create, calculate, migrate, activate, etc.
  - b) **weak verbs**: make, report, use, organize, record, etc.
- 2) a weak noun may refer to several objects, a strong noun clearly identifies only one object
  - a) **strong nouns**: property, payment, transcript, etc.
  - b) **weak nouns**: data, paper, report, system, etc.

e-Macao-18-1-192

## Naming Actors

- 1) group individuals according to how they use the system by identifying the roles they adopt while using the system
- 2) each role is a potential actor
- 3) name each role and define its distinguishing characteristics
- 4) do not equate job titles with roles; roles cut across jobs
- 5) use common names for existing system; avoid inventing new

e-Macao-18-1-193

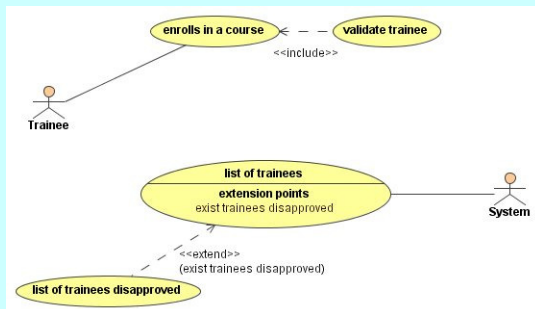
## Use Case Template

Fields	Description
Use Case Name	Name of the use case
Actors	Role names of people or external entities initiating the use case
Purpose	The intention of the use case
Overview	A brief description of the usage of the process
Precondition	A condition that must hold before a use case can begin
Variation	Different ways to accomplish use case actions
Exceptions	What might go wrong during the execution of the use case
Policies	Specific rules that must be enforced by the use case
Post-conditions	Condition that must prevail after executing the use case
Priority	How important is the use case?
Frequency	How often is the use case performed?
Cross reference	Relate use cases and functional requirements

e-Macao-18-1-194

## Task 25

Is this diagram correct ? If not, explain.

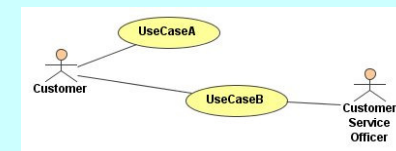


e-Macao-18-1-195

## Task 26

In the following bank ATM-related use cases, what would best describe a use case of type *UseCaseA*?

- a) Open Account
- b) Withdraw from Account
- c) Close Account
- d) Reopen Account
- e) None of the above



### A.4.3. Conceptual Modelling

## Requirements Modelling

### Class and Object Diagrams

e-Macao-18-1-197

## Course Outline

1) Object Orientation

2) UML Basics

3) UML Modelling:a) Requirements

b) Architecture

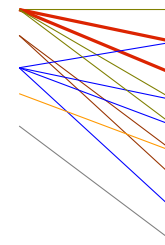
c) Design

d) Implementation

e) Deployment

4) Unified Process

5) UML Tools



UML Diagrams:

1. use case

2. class3. object

4. sequence

5. state

6. component

7. collaboration

8. activity

9. deployment

e-Macao-18-1-198

## Concept Definition

What is a concept? An **idea**, **thing** or **object**.

An object can be:

- 1) represented symbolically
- 2) defined or described
- 3) exemplified

What is an instance? Each concrete application of the concept.

e-Macao-18-1-199

## Concept Examples

- 1) agency
- 2) license
- 3) officer
- 4) applicant
- 5) internal applicant
- 6) external applicant

e-Macao-18-1-200

## Concept Identification

Concepts can be:

- 1) physical or tangible objects
- 2) places
- 3) documents, specifications, design or descriptions
- 4) roles of people
- 5) container of other things
- 6) organizations
- 7) processes
- 8) catalogs, etc.

Concepts are identified through requirements and use cases.

e-Macao-18-1-201

## Conceptual Model and Classes

**Conceptual model:**

- 1) captures the concepts in a domain in an abstract way
- 2) important part of OO requirements analysis

**Classes:**

- 1) equivalent to concepts in UML
- 2) an abstraction of a set of objects
- 3) objects are concrete entities existing in space and time

e-Macao-18-1-202

## Class Diagrams

- 1) the most widely used diagram of UML
- 2) models the static design view of a system
- 3) also useful in modelling business objects
- 4) used to specify the structure, interfaces and relationships between classes that underlie the system architecture.
- 5) primary diagram for generating codes from UML models

e-Macao-18-1-203

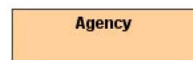
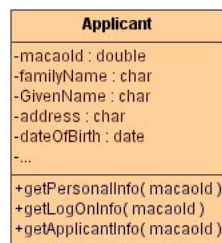
## A Class

A class is:

- 1) a description of a set of objects that share the same
  - a) attributes,
  - b) operations,
  - c) relationships and
  - d) semantics
- 2) a software unit that implements one or more interfaces

e-Macao-18-1-204

## Class Examples



e-Macao-18-1-205

## Class Notation

Basic notation: a solid-outline rectangle with three compartments separated by horizontal lines.

Three compartments:

- 1) top compartment holds the class name and other general properties of the class
- 2) middle compartment holds a list of attributes
- 3) bottom compartment holds a list of operations

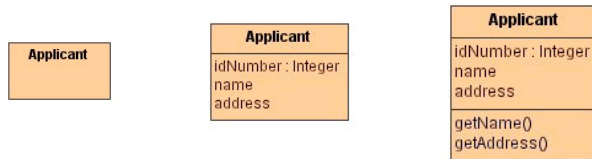
Alternative styles:

- 1) suppress the attributes compartment
- 2) suppress the operation compartment



e-Macao-18-1-206

## Class Notation Example



e-Macao-18-1-207

## Stereotypes

- 1) a mechanism allowing to extend the semantics of UML
- 2) used to present more information about an artifact
- 3) notation – the name of a new element within the matched guillemets, e.g. <<thread>>

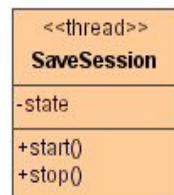
e-Macao-18-1-208

## Stereotypes for Classes

Attribute or operation lists in a class may be organized into groups with **stereotypes**.

A number of stereotypes of classes and data types:

- 1) thread
- 2) event
- 3) entity
- 4) process
- 5) utility
- 6) metaclass
- 7) powerclass
- 8) enumeration, etc.



e-Macao-18-1-209

## Stereotype Descriptions

Stereotype	Description
Thread	An active class which specifies a lightweight flow that can execute concurrently with other threads within the same processes.
Process	An active class which specifies a heavyweight flow that can execute concurrently with other processes.
Control	A class which owns almost no information about itself. It represents a behaviour rather than resources and directs the behaviour of other objects almost having no behaviour of its own.
Entity	A class which represents a resource in the real world. It describes its features and their current condition (their state) and preserves its own integrity regardless of where and when it is used.
Utility	A class whose attributes and operations are all class scoped. That is a class which no instance.
Metaclass	A classifier whose objects are all classes.
Powerclass	A classifier whose objects are the children of a given parent.
Enumeration	A user defined data type that defines a set of values that do not change.

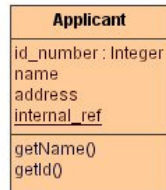
e-Macao-18-1-210

## Attribute/Operation Scope

Different scopes can be specified for the attributes and operations (features) of a class:

- 1) **instance scope** - a feature appears in each instance of the class (classifier)
- 2) **class scope** - there is just a single instance of the feature for all instances of a class (classifier)

Underlining the feature's name indicates the classifier scope.



e-Macao-18-1-212

## Class Relationships

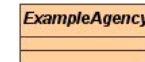
Four relationships between classes:

- 1) association
- 2) aggregation
- 3) generalization
- 4) dependency

e-Macao-18-1-211

## Types of Classes

- 1) **abstract class**
  - a) cannot have direct instances
  - b) the name is written in italics
- 2) **root class**
  - a) cannot be a sub-class
  - b) written with *root* stereotype
- 3) **leaf class**
  - a) cannot be a super-class
  - b) written with *leaf* stereotype



e-Macao-18-1-213

## Relationship 1: Association

Association - a relationship between two or more classifiers that involves connection among instances.

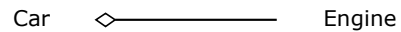


Example: Person works for the Company.

e-Macao-18-1-214

## Relationship 2: Aggregation

Aggregation - A special form of association that specifies a whole-part relationship between the aggregate (whole) and the component part.



Example: Car has an Engine.

e-Macao-18-1-215

## Relationship 3: Generalization

Generalization - A taxonomic relationship between a more general and a more specific element.



Example: Truck is a Vehicle.

e-Macao-18-1-216

## Relationship 4: Dependency

Dependency - A relationship between two elements, in which a change to one element (source) will affect another (target).

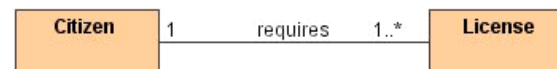


Example: Procedure depends on the Policy.

e-Macao-18-1-217

## Multiplicities for Classes

Shows how many objects of one class can be associated with one object of another class



Example: a citizen can apply for one or more licenses, and a license is required by one citizen.

e-Macao-18-1-218

## Multiplicities Syntax

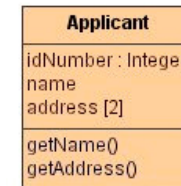
Value	Description
0..0	Zero
0..1	Zero or one
0..*	Zero or more
1..1	One
1..*	One or more
*	Unlimited number
<literal>	Exact number (Example: 4)
<literal>..*	Exact number or more (Example: 4..* indicating 4 or more)
<literal>..<literal>	Specified range (Example: 4..13)
<literal>..<literal>, <literal>	Specified range or exact number (Example: 4..13,31 indicating 4 through 13 and 31)
<literal>..<literal>,<literal>..<literal>	Multiple specified ranges (Example: 4..13, 31-41)

e-Macao-18-1-219

## Multiplicities for Attributes

Can specify how many instances of an attribute can be associated with one instance of the class.

Example: an applicant can have two addresses.



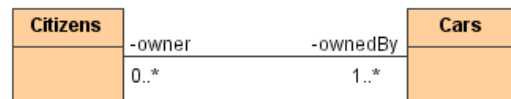
e-Macao-18-1-220

## Roles

A **role** names a behaviour of an entity participating in a particular relationship.

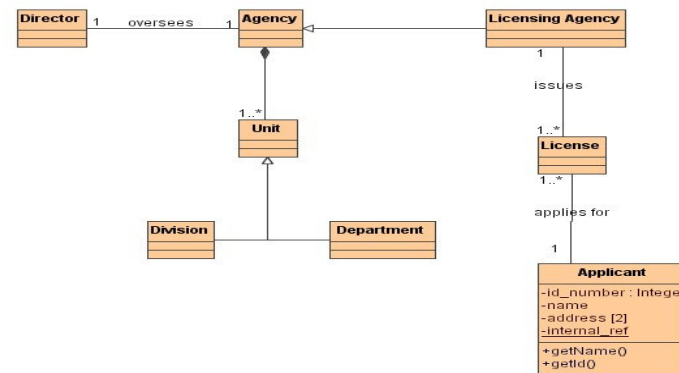
Notation: add the name of the role at the end of the association line, next to the class icon.

Example: a citizen own a car, sell cars, be an employee, etc.



e-Macao-18-1-221

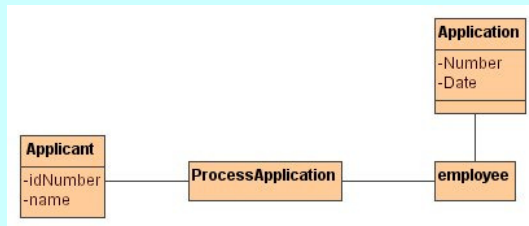
## Class Diagram Example



e-Macao-18-1-222

## Task 27

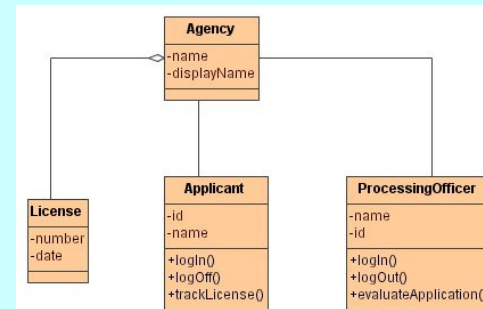
Is this diagram correct ? If not, explain.



e-Macao-18-1-223

## Task 28

Discover and correct all mistakes in the following diagram :



e-Macao-18-1-224

## Task 29

Using a UML tool produce a class diagram with all the classes and relationships defined in the Tasks 6, 8 and 10.

e-Macao-18-1-225

## Object Diagram

- 1) models the instances of classes contained in class diagrams
- 2) shows a set of objects and their relationships at one time
- 3) modelling object structures involves taking a snapshot of a system at a given moment in time
- 4) is an instance of a class diagram or the static part of an interaction diagram
- 5) it contains objects and links

e-Macao-18-1-226

## Object Diagram Usage

Object diagrams are used to:

- 1) visualize
- 2) specify
- 3) construct
- 4) document

The existence of certain instances in a system, together with their relationships.

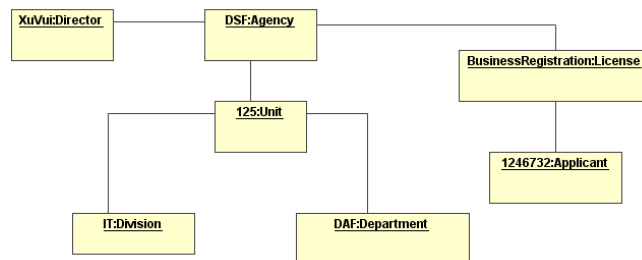
e-Macao-18-1-227

## Creating an Object Diagram

- 1) identify the function/behaviour of interest that results from interaction of classes, interfaces and other artifacts
- 2) for each function/behaviour, identify the artifacts that participate in the collaboration as well as their relationships
- 3) consider one scenario that invokes the function/behaviour, freeze the scenario and render each participating object
- 4) expose the state and attribute values of each object, as necessary to understand the scenario
- 5) expose the links among these objects

e-Macao-18-1-228

## Object Diagram Example



e-Macao-18-1-229

## Task 30

Using a UML tool produce an object diagram illustrating the class diagram shown in the previous task.

e-Macao-18-1-230

## Task 31

---

Select the best answer:

An object is named:

- a) with a noun
- b) with a noun, a colon and then a class name
- c) with a number and a colon followed by a class name
- d) just like a class

e-Macao-18-1-231

## Task 32

---

Select the best answer:

The object diagram is used for:

- a) testing and verifying the use cases
- b) modelling scenarios
- c) analyzing and testing class diagrams

#### A.4.4. Behavioural Modelling

## Requirements Modelling

### Sequence and State Diagrams

e-Macao-18-1-233

## Course Outline

1) Object Orientation

2) UML Basics

3) UML Modelling:a) Requirements

b) Architecture

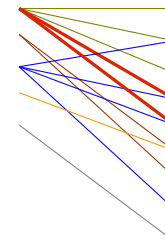
c) Design

d) Implementation

e) Deployment

4) Unified Process

5) UML Tools



UML Diagrams:

1. use case

2. class

3. object

4. sequence5. state

6. component

7. collaboration

8. activity

9. deployment



e-Macao-18-1-234

## Behavioral Diagrams 1

- 1) represent how objects behave when you put them to work using the structure already defined in structural diagrams
- 2) model how the objects communicate in order to accomplish tasks within the operation of the system
- 3) describe how the system:
  - a) responds to actions from the users
  - b) maintains internal integrity
  - c) moves data
  - d) creates and manipulates objects, etc.

e-Macao-18-1-235

## Behavioral Diagrams 2

- 4) describe discrete pieces of the system, such as individual **scenarios** or operations

**Note:** not all system behaviour have to be specified - simple behaviours may not need a visual explanation of the communication required to accomplish them.

e-Macao-18-1-236

## Scenario

### Definition

**Scenario** is a textual description of how a system behaves under a specific set of circumstances.

Behaviour described in use cases is the basis for scenarios.

Scenarios also provide a basis for developing test cases and acceptance-level test plans.

e-Macao-18-1-237

## Scenario Example

Consider the use case "Track License Application"

There are at least two possible scenarios:

- a) the applicant enters the license application number; the system retrieves the information related to it; the system displays this information
- b) the applicant enters the license application number; the number does not exist in the agency's database; the system displays an error message

e-Macao-18-1-238

## Scenario Example

**Scenario:** the applicant enters the license application number; the system retrieves the information related to it; the system displays this information.

**Steps:**

- 1) Applicant requests to track status of a license application
- 2) System displays the logon form
- 3) Applicant enters the logon information
- 4) Applicant submits the logon information
- 5) System validates the applicant
- 6) System displays the form to enter the tracking number
- 7) Applicant enters the tracking number
- 8) Applicant submits the license number
- 9) System retrieves the license information
- 10) System displays the license information

e-Macao-18-1-239

## Task 33

Select a use case among those defined on Task 22 and define a possible scenario for it.

e-Macao-18-1-240

## Task 34

Select the best answer:

Scenarios are:

- a) the same as use cases
- b) the same as test cases
- c) used to derive test cases
- d) the same as object diagrams

e-Macao-18-1-241

## Sequence Diagram

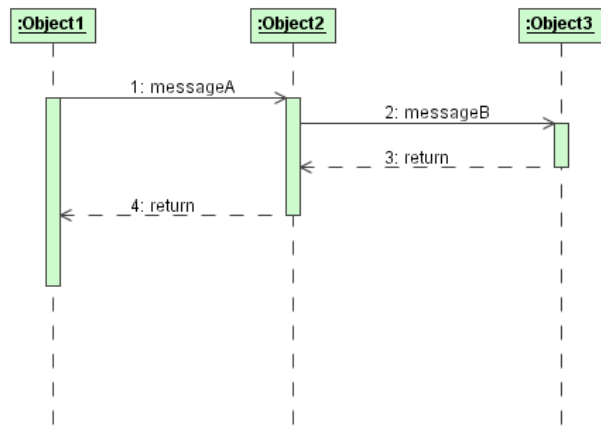
A sequence diagram shows interactions between objects.

Components of sequence diagrams:

- 1) object lifelines
  - a) object
  - b) timeline
- 2) messages
  - a) message, stimulus
  - b) signal, exception
  - c) operations, returns
  - d) identification
- 3) message syntax

e-Macao-18-1-242

## Sequence Diagram Example



e-Macao-18-1-243

## Timeline

The *timeline* is a line that runs:

1. from the beginning of a scenario at the top of the diagram
2. to the end of the scenario at the bottom of the diagram.

e-Macao-18-1-244

## Object Lifeline

An object lifeline consists of:

- 1) an object
- 2) a timeline

If an object is created and destroyed during the message sequence, the lifeline represents its whole lifecycle.



e-Macao-18-1-245

## Message

### Definition

**Message** is a description of some type of communication between objects.

A unit of communication between objects.

The sender object may:

- 1) invoke an operation,
- 2) raise a signal or
- 3) cause the creation or destruction of the target object.

e-Macao-18-1-246

## Message Notation

Notation: modelled as an arrow where the tail of the arrow identifies the sender and the head points to the receiver.

sender            receiver

For each message we need to define:

- 1) the name of the invoked operation and its parameters
- 2) the information returned from the message

e-Macao-18-1-247

## Stimulus

### Definition

**Stimulus** is an item of communication between two objects:

- 1) it is associated with both sending and receiving objects
- 2) it travels across a link
- 3) it may:
  - a) invoke an operation,
  - b) raise a signal (asynchronous message),
  - c) create or destroy an object
- 4) it may include parameters/arguments in the form of primitive values or object references
- 5) it is associated with a procedure that causes it to be sent

e-Macao-18-1-248

## Stimulus Example

### 1) Example 1 – invoke an operation

The citizen object submits the application form by sending a message to the system. The system receives the message and executes the associated procedure.

### 1) Example 2 – create an object

When the System receives the message from the applicant submitting the application form, it creates an object to support the new session.

e-Macao-18-1-249

## Messages and Stimuli

A **message** is the **specification** of a **stimulus**.

The specification includes:

- 1) the roles the sender/receiver objects play in the interaction
- 2) the procedure that dispatches the stimulus

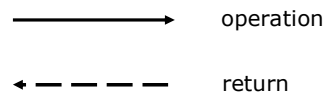
A stimulus is an instance of a message.

e-Macao-18-1-250

## Operations and Returns

The **operation** specifies the procedure that the message invokes on the receiving object.

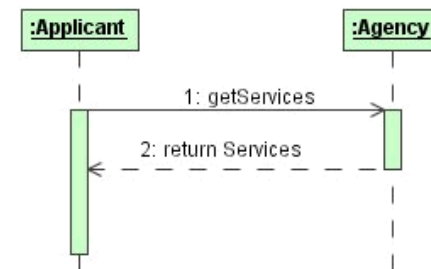
The **return** contains the information passed back from the receiver to the sender. An empty return is valid.



e-Macao-18-1-251

## Operations/Returns Example

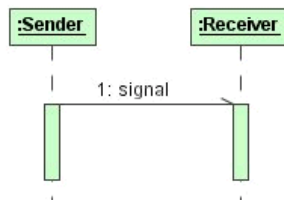
Example: The *Applicant* object sends a message to the *Agency* object to get the list of services it provides. The *Agency* object returns this information.



e-Macao-18-1-252

## Signal

- 1) an object may raise a signal through a message
- 2) a **signal** is a special type of a class associated with an event that can trigger a procedure within the receiving object
- 3) a signal does not require a return from the receiving object



e-Macao-18-1-253

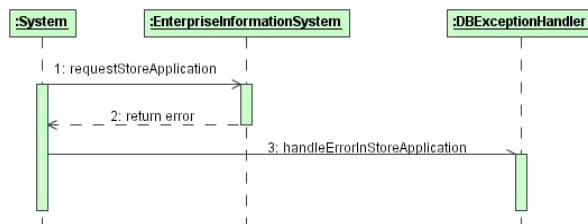
## Exception

- 1) an **exception** is a special type of a signal
- 2) **throwing an exception** means sending out a message containing an object that describes the error condition

e-Macao-18-1-254

## Signal Example

Each time System receives a message indicating that some abnormal situation occurred with the database, it raises a signal to DBExceptionHandler to handle the error.

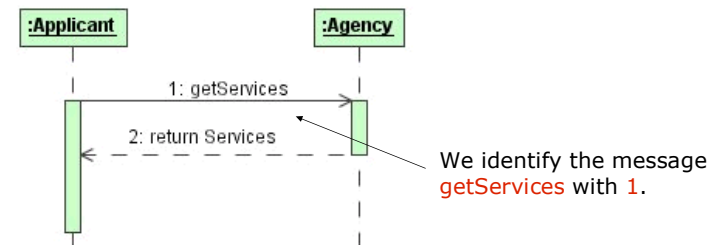


e-Macao-18-1-255

## Identification of Messages

A message number or name is used to identify messages.

Example:



e-Macao-18-1-256

## Message Syntax 1

All former concepts – messages, signals, operations, returns, etc. are part of the message syntax.

*predecessors* '/' *sequence-term*  
*iteration* [*condition*] *return* ':' *operation*

where:

- 1) *predecessors* is a comma-separated list of sequence numbers of all messages that must come before the current message
- 2) *sequence-term* may be either a number or a name that identifies the message

e-Macao-18-1-257

## Message Syntax 2

- 3) *iteration* determines if a message should be sent once or several times in a sequence:
  - a) one message - add an iteration symbol (\*) and a condition to control the number of iterations
  - b) many messages - enclose the set of messages in a box
- 4) *condition* specifies the control of the iteration; expressed as a text enclosed within square brackets
- 5) *return* may include a list of values sent back to the sender
- 6) *operation* defines the name of the operation and optionally its parameters and a return value

e-Macao-18-1-258

## Message Syntax Example

Example:

```
6/8:getAddress * [foreach ApplicationForm]
return text:= getAddress(Citizen.Id:Integer)
```

1. Specifies the message number 8 called getAddress.
2. The message will be executed more than once (\*), one time for each ApplicationForm.
3. Each message calls the operation getAddress of the receiving object, sending CitizenId parameter of type Integer, and returns a value of type text.
4. For the execution of this message, it is required that the message 6 has already been executed.

e-Macao-18-1-259

## Example Scenario

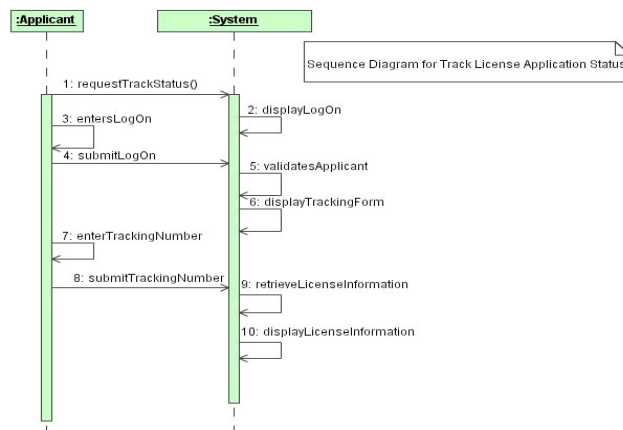
Recall the scenario: an applicant tracks the status of a license application and the system displays the license information.

**Procedure:**

1. Applicant requests to track the status of a license application
2. System displays the logon form
3. Applicant enters the logon information
4. Applicant submits the logon information
5. System validates the applicant
6. System displays the form to enter the tracking number
7. Applicant enters the tracking number
8. Applicant submits the tracking number
9. System retrieves the license information
10. System displays the license information

e-Macao-18-1-260

## Example Sequence Diagram



e-Macao-18-1-261

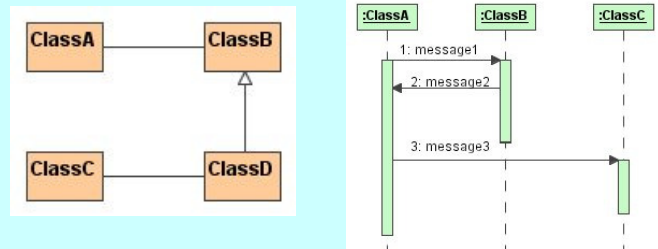
## Task 35

- Based on the scenario defined in the previous Task, produce the sequence diagram using the UML tool.

e-Macao-18-1-262

## Task 36

Is there any problem in the following sequence diagram? Explain.



e-Macao-18-1-263

## Task 37

Select the best answer:

A timeline is:

- a) an event signaling the termination of a timed process, much the same as an alarm on a timer
- b) another name for a sequence
- c) used in a sequence diagram as an alternative to numbering events
- d) the amount of time it takes to complete a set of iterations

e-Macao-18-1-264

## Statechart Diagrams

A statechart diagram defines the behaviour of a single object or of a set of objects related by a collaboration.

It captures the changes in an object throughout its lifecycle as they occur in response to internal and external events.

The scope of a statechart is the entire life of one object.

e-Macao-18-1-265

## Statechart Components

Statecharts are composed of:

- 1) states
  - a) initial state
  - b) final state
- 2) events
  - a) guard conditions
  - b) actions
  - c) event syntax
- 3) complex states
  - a) activities
  - b) entry actions
  - c) exit actions



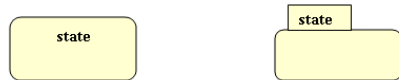
e-Macao-18-1-266

## State

### Definition

**State** is the current condition of an object reflected by the values of its attributes and its links to other objects.

Notation:



Particular states are initial and final states.

e-Macao-18-1-267

## Initial State

The **initial state** identifies the state in which an object is created or constructed.

The initial state is called a **pseudo-state** because it does not really have the features of an actual state, but it helps clarify the purpose of another state of the diagram.

Notation:



e-Macao-18-1-268

## Final State

The **final state** is the state in which once reached, an object can never do a transition to another state.

The final state may also mean that the object has actually been destroyed and can no longer be accessed.

Notation:



e-Macao-18-1-269

## Event

### Definition

**Event** is an occurrence of a stimulus that can trigger a state transition.

An event may be:

- 1) the **receipt of a signal**, e.g. the reception of an exception
- 2) the **receipt of a call**, that is the invocation of an operation, e.g. for changing the expiration date of a license

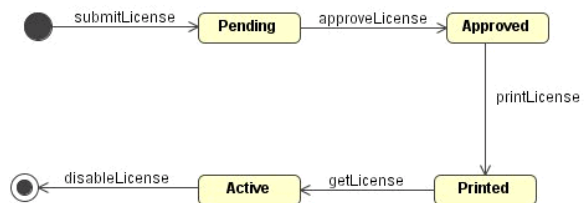
An event on a statechart diagram corresponds to a message on a sequence diagram.

Notation:



e-Macao-18-1-270

## Statechart Diagram Example

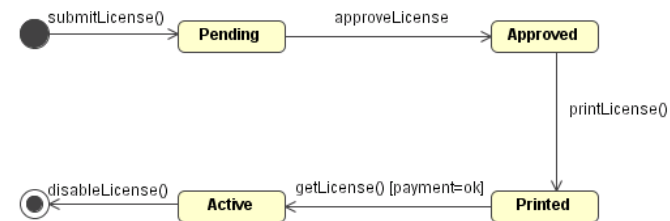


e-Macao-18-1-271

## Guard Condition

Typically, an event is received and responded unconditionally.

When the receipt of an event is conditional, the test needed is called the guard condition.



e-Macao-18-1-272

## Event Actions

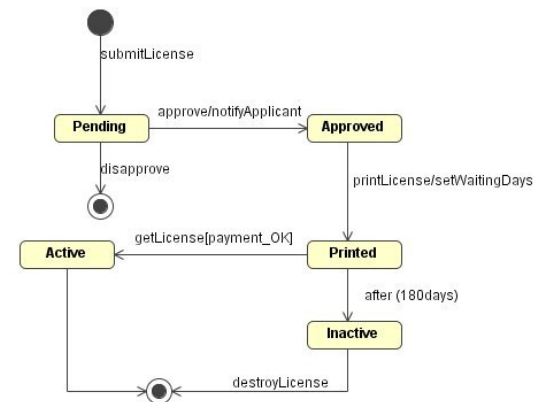
The response to an event has to explain how to change the attribute values that define the object's state.

The behaviour associated with an event is called **action expression**:

1. Part of a transition event – specifying the change from one state to another.
2. An atomic model of execution, referred to as run-to-completion semantics.

e-Macao-18-1-273

## Event Actions Example



e-Macao-18-1-274

## Event Syntax

event-name '(' [comma-separated-parameters-list] ')' '  
 '['guard-condition']' / [action-expression]

where:

- 1) **event-name** - identifies the event
- 2) **parameters-list** - data values passed with the event for use by the receiving object in its response to the event
- 3) **guard-condition** - determines whether the receiving object should respond to the event
- 4) **action-expression** - defines how the receiving object must respond to the event

e-Macao-18-1-275

## Event Syntax Example

Example:

approveLicense(License.Id) [req=ok]  
 /setExistLicense(true)

where:

- 1) approveLicense is the event name
- 2) License.Id is the event parameter
- 3) the guard condition specifies that the req attribute must be OK for the receiving object to respond to the event.
- 4) the action executed in the receiving object is a call to the method setExistLicense which sends true as its parameter

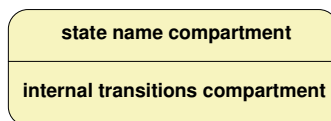
e-Macao-18-1-276

## Complex States

The state icon can be expanded to model what the object can do while it is in a given state.

The notation splits the state icon into two compartments:

1. name compartment and
2. internal transitions compartment.



Internal transitions compartments contains information about actions, activities and internal transitions specific to that state.

e-Macao-18-1-277

## Entry Actions

More than one event can trigger a transition of an object into the same state.

When the same action takes place in all events that goes into a state, the action may be written once as entry action.

Notation:

- 1) use the keyword **entry** followed by a slash and the actions to be performed every time the state is entered
- 2) entry actions are part of internal transitions compartment

e-Macao-18-1-278

## Exit Actions

The same simplification can be used for actions associated with events that trigger a transition out of a state.

They are called exit actions.

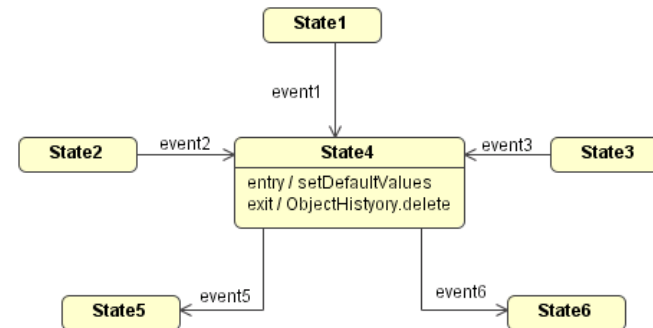
Notation:

- 1) use the keyword exit followed by a slash and the actions performed every time the state is exited
- 2) exit actions are part of the internal transitions compartment

Only when the action takes places every time the state is exited.

e-Macao-18-1-279

## Entry/Exit Actions Example



e-Macao-18-1-280

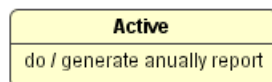
## Activities 1

Activities are the processes performed within a state.

Activities may be interrupted because they do not affect the state of the object.

Notation:

- 1) use the do keyword followed by a slash and activities
- 2) activities are placed in the internal transitions compartment.



e-Macao-18-1-281

## Activities 2

An activity should be performed:

- 1) from the time the object enters the state
- 2) until
  - a) either the object leaves the state or
  - b) the activity finishes.

If an event produces a transition out of the activity state, the object must shut down properly and exit the state.

e-Macao-18-1-282

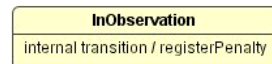
## Internal Transitions

An event that can be handled completely without a change in state is called an **internal transition**.

It also can specify guard conditions and actions.

Notation:

- 1) uses the keyword **internal transition** followed by a slash and one event action
- 2) they are placed in the internal transitions compartment



e-Macao-18-1-283

## Order of Events

- 1) if an activity is processed in the current state, interrupt it and finish it properly
- 2) execute the exit actions
- 3) execute the actions associated with the event that triggered the transition
- 4) execute the entry actions of the new state
- 5) begin executing the activities of the new state

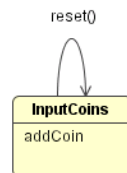
e-Macao-18-1-284

## Self Transition

A self-transition is an event that is sufficiently significant to interrupt what the object is doing.

It forces the object to exit the current state and return to the same state.

The result is to stop any activity within the object, exit the current state and re-enter the state.



e-Macao-18-1-285

## Important Features

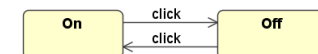
Within the statechart diagram:

- 1) an object need not know who sent the message
- 2) an object is only responsible for how it responds to the event

Focusing on the condition of the object and how it responds to the events, which object sends the message becomes irrelevant and the model is simplified

The state of the object when it receives an event can affect the object's response

**event + state = response**



e-Macao-18-1-286

## Defining Send Events

Sometimes the object modelled by the statechart needs to send a message to another object, in this case the outgoing event must define which is the receiving object. Also, this event must be caught by the receiving object.

A message send to another object is called a **send event**.

Notation: provide the object name before the action expression with a period separating both.



e-Macao-18-1-287

## Relating Diagrams 1

- 1) the sequence diagram models the interactions between objects
- 2) the statechart diagram models the effect that these interactions have on the internal structure of each object
- 3) the messages modelled in the sequence diagrams are the external events that place demands on objects

e-Macao-18-1-288

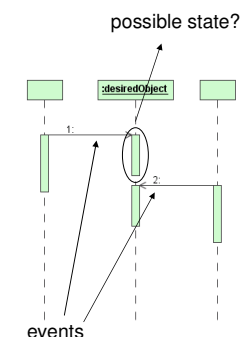
## Relating Diagrams 2

- 4) the objects internal responses to those events that cause changes to the objects' states are represented in the statechart diagram
- 5) not all objects need to be modelled with a statechart diagram
- 6) the objects that appear in many interactions and are target of many events are good candidates to be modelled with a statechart diagram

e-Macao-18-1-289

## Sequence to Statechart

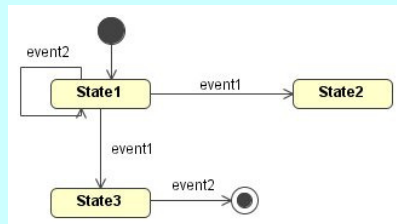
- 1) identify the events directed at the lifeline of the desired object
- 2) identify candidate states by isolating the portions of the lifeline between the incoming events
- 3) name the candidate states using adjectives that describe the condition of the object during the period of time represented by the gap
- 4) add the new state and events to the statechart diagram



e-Macao-18-1-290

## Task 38

Discover and correct all mistakes in the following diagram:



e-Macao-18-1-291

## Task 39

Select the best answer:

A state is the condition of an object

- a) upon construction
- b) that governs whether an event will trigger a transition
- c) at the beginning and at the end of a scenario
- d) at a point in time

e-Macao-18-1-292

## Task 40

Select the best answer:

An event action is an action that:

- a) must take place when an object enters the state
- b) must take place when triggered by the associated event
- c) must always take place when an object leaves the state
- d) causes no change in an object's state

e-Macao-18-1-293

## Task 41

- Using a UML tool define a statechart diagram for the license class. The diagram should contain at least three states.
- What is the relation between this diagram and the class diagram defined in the conceptual modelling?
- What is the relation between this diagram and the set of sequence diagrams?

e-Macao-18-1-294

## Summary 1

A requirement is a function that a system must perform or a desirable characteristic of a system.

There are different kinds of requirements such as functional and non-functional requirements.

Most project failures can be traced back to errors made in the requirements gathering and specification.

e-Macao-18-1-295

## Summary 2

Use cases are descriptions of sets of action sequences that a system performs to deliver observable results.

Use cases may be related using generalization, include and extend relationships.

Actors are the entities that interact with the system, causing it to respond to events.

Use case diagrams show a set of:

1. use cases,
2. actors and
3. their relationships.

e-Macao-18-1-296

## Summary 3

Conceptual modelling helps to understand application domains.

Conceptual class diagrams describe and relate the concepts of a domain. They show attributes but not methods.

An object diagram models the instances of the classes contained in a given class diagram.

e-Macao-18-1-297

## Summary 4

Behavioral modelling specifies how objects work together to provide a specific behaviour.

Sequence diagrams show how objects interact in order to deliver a discrete piece of the system functionality.

Statechart diagrams capture the changes in an object or a set of related objects as they occur in response to events.



e-Macao-18-1-298

## Summary 5

Sequence diagrams:

- 1) show interactions between objects.
- 2) model the dynamic behaviour of a system.
- 3) emphasize the chronological ordering of messages.

e-Macao-18-1-299

## Summary 6

Statechart diagrams:

- 1) show a state machine consisting of states, transitions, events and activities.
- 2) model the dynamic behaviour of a system.
- 3) emphasize the event-driven ordering.

## A.5. Architecture

### A.5.1. Software Architecture

# Architecture Modelling

e-Macao-18-1-301

## Course Outline

1) Object Orientation

2) UML Basics

3) UML Modelling:

a) Requirements

b) Architecture

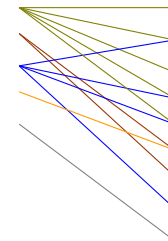
c) Design

d) Implementation

e) Deployment

4) Unified Process

5) UML Tools



UML Diagrams:

1. use case

2. class

3. object

4. sequence

5. state

6. component

7. collaboration

8. activity

9. deployment

e-Macao-18-1-302

## Architecture

Architecture involves defining:

- a) **what** are the **main components** of the system
- b) **how** this components **are related**.

The architecture shows:

- a) the structural organization of a system from its components
- b) how elements interact to provide the system's overall behaviour or required functionality

Architectural concerns:

- a) structural or static
- b) behavioural

e-Macao-18-1-303

## Architecture Definition

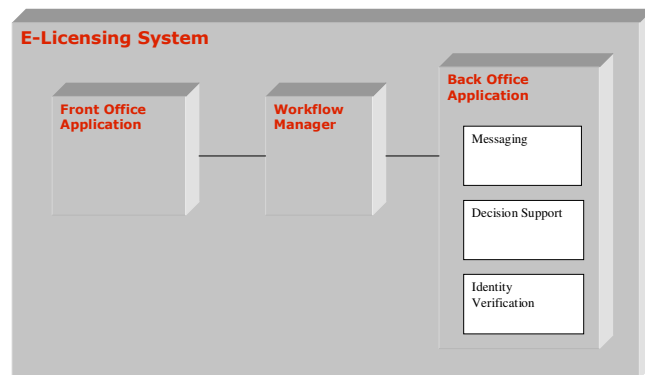
### Definition

**Architecture** is a set of **significant decisions** about the organization of a software system. Such decisions include:

1. the **selection of structural elements** and their **interfaces**
2. the **composition** of these structural and behavioural element into progressively larger subsystem
3. the **architectural style** that guides this organization – the elements and their interfaces, their collaboration and composition

e-Macao-18-1-304

## Architecture Example



e-Macao-18-1-305

## Architecture Concepts

Some concepts related to architecture:

- 1) subsystems
  - a) classes
  - b) services
- 2) design principles for defining subsystems:
  - 1) coupling
  - 2) cohesion
- 3) layering strategy for defining subsystems:
  - 1) responsibility driven
  - 2) reuse driven

e-Macao-18-1-306

## Subsystems: Classes

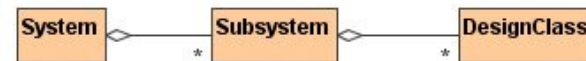
A solution domain may be decomposed into smaller parts called **subsystems**.

Subsystems may be recursively decomposed into simpler subsystems.

Subsystems are composed of solution domain classes (design classes).

e-Macao-18-1-307

## Subsystems: Classes Example



e-Macao-18-1-308

## Subsystems: Services

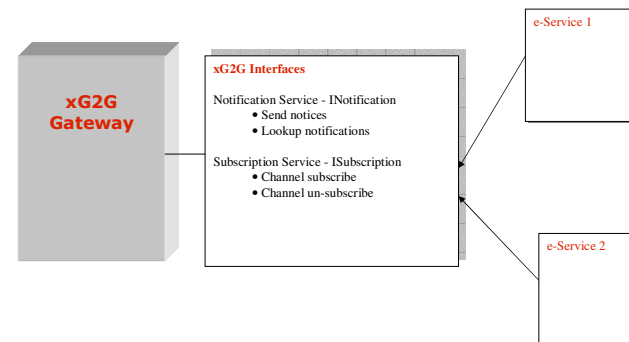
A subsystem is characterized by the **services** it provides to other subsystems.

A **service** is:

- 1) a set of related operations that share a common purpose
- 2) a set of operations of a subsystem that are available to other subsystem through the **subsystem's interface**

e-Macao-18-1-309

## Subsystem: Services Example



e-Macao-18-1-310

## Coupling

### Definition

**Coupling** is the strength of dependencies between two sub-systems.

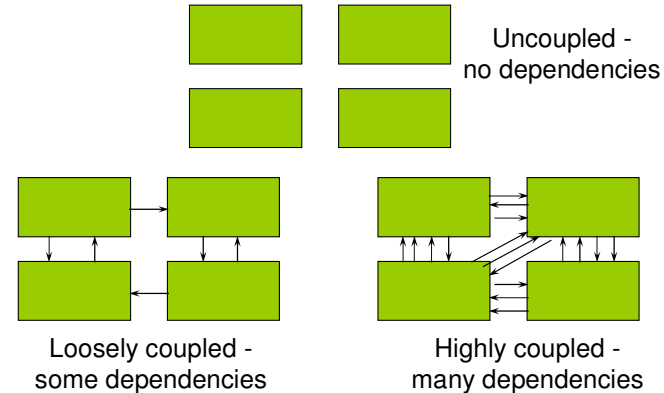
Loose coupling results in:

- 1) sub-system independence
- 2) better understanding of sub-systems
- 3) easier modification and maintenance

High coupling is generally undesirable.

e-Macao-18-1-311

## Coupling Example



e-Macao-18-1-312

## Cohesion

### Definition

**Cohesion** or **Coherence** is the strength of dependencies within a subsystem.

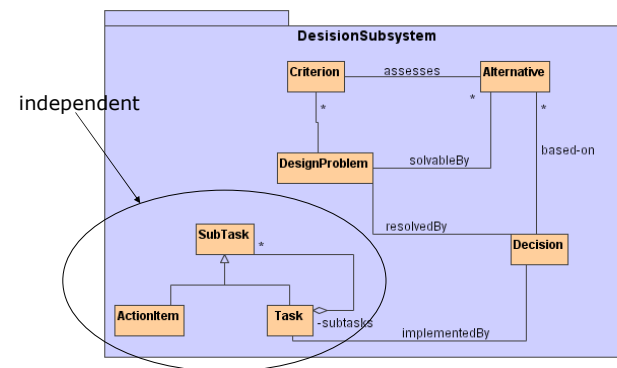
In a highly cohesive subsystem:

- subsystem contains related objects
- all elements are directed toward and essential for performing the same task.

Low cohesion is generally undesirable

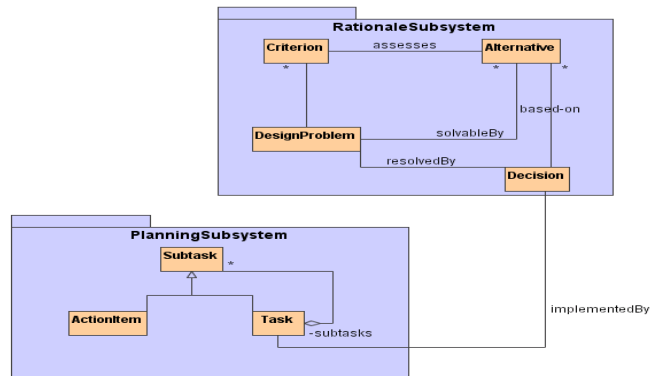
e-Macao-18-1-313

## Low Cohesion Example



e-Macao-18-1-314

## High Cohesion Example



e-Macao-18-1-315

## Layering

Layering is a strategy for dividing system into subsystems.

Layering:

- 1) divides a system into a hierarchy of subsystems
- 2) follows two common approaches:
  - a) responsibility driven
  - b) reuse driven

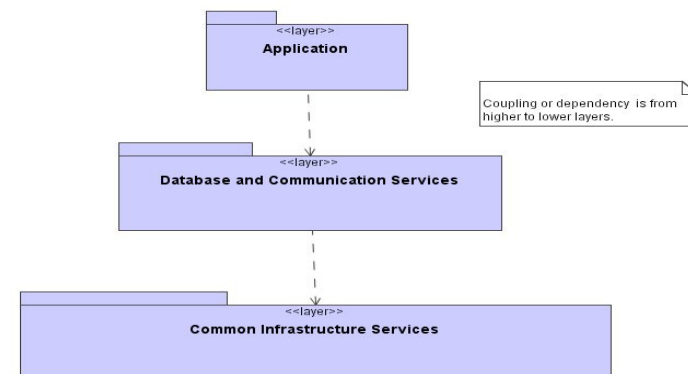
e-Macao-18-1-316

## Layering styles

- 1) **responsibility driven:**
  - a) layers have well-defined responsibilities
  - b) layers fulfill specific roles
- 2) **reuse driven:**
  - a) layers are designed to allow maximum reuse of system elements
  - b) higher level layers use services of lower level layers

e-Macao-18-1-317

## Layered Architecture Example



e-Macao-18-1-318

## Task 42

---

- 1) Consider a typical client-server application. Assume that the client subsystem is a layer and the server is the other layer. Which of the two layering styles best describes this architecture?
- 2) Justify your answer in question 1.

### A.5.2. Collaboration Diagrams

## Architecture Modelling

### Collaboration Diagrams

e-Macao-18-1-320

## Course Outline

1) Object Orientation

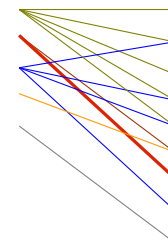
2) UML Basics

3) UML Modelling:

- a) Requirements
- b) Architecture
- c) Design
- d) Implementation
- e) Deployment

4) Unified Process

5) UML Tools



UML Diagrams:

- 1. use case
- 2. class
- 3. object
- 4. sequence
- 5. state
- 6. component
- 7. collaboration
- 8. activity
- 9. deployment



e-Macao-18-1-321

## Collaboration

### Definition

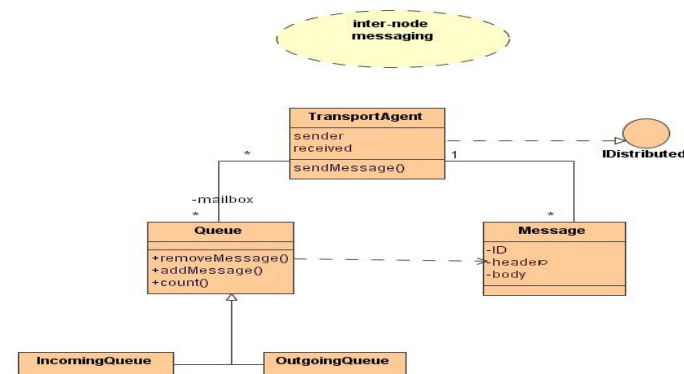
A **collaboration** is a society of classes, interfaces, and other elements that work together to deliver or provide some cooperative behaviour that is bigger than the sum of all its parts.

A collaboration:

- 1) names a **conceptual** chunk that encompasses both static and dynamic aspects
- 2) specifies the realization of a use case

e-Macao-18-1-322

## Collaboration Example



*Inter-node Messaging Collaboration and its details*

e-Macao-18-1-323

## Collaboration Names

- 1) every collaboration must have a name
- 2) collaboration names are nouns or short noun phrases, typically first letter of the letter is capitalized.

Example: "Inter-node Messaging" or "Application Submission"

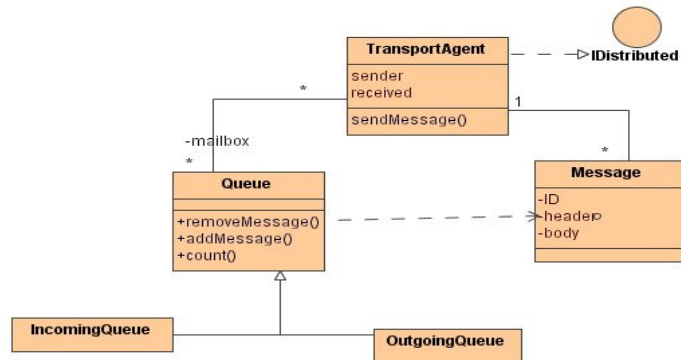
e-Macao-18-1-324

## Collaboration – Structural

- 1) specifies the classifiers, such as classes, interfaces, components and nodes that are required to interact
- 2) does not own any of its structural elements
- 3) only references the classifiers declared elsewhere

e-Macao-18-1-325

## Collaboration Structural View



*Structural View of Collaboration which implements a "send and receive message" use case*

e-Macao-18-1-326

## Task 43

Select one of the use cases already defined restaurant licensing application:

- 1) identify the classes that are essential in realizing this use case
- 2) show the relationship between these classes
- 3) name the collaboration defined by these classes

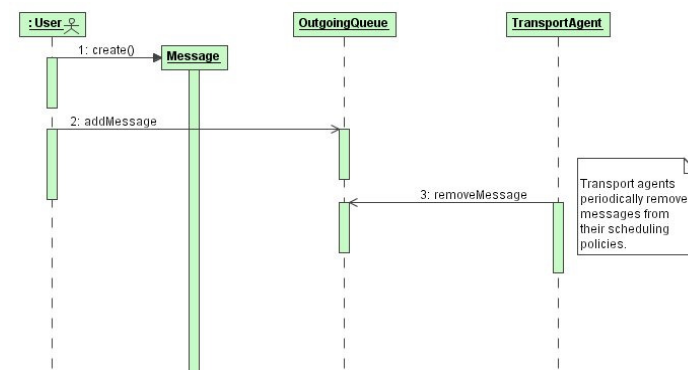
e-Macao-18-1-327

## Collaboration – Behavioural

- 1) rendered using interaction diagram
- 2) specifies a set of messages that are exchanged among a set of objects to accomplish a specific purpose – a use case or operation
- 3) defines an interaction context

e-Macao-18-1-328

## Collaboration Behavioral View



e-Macao-18-1-329

## Collaboration Diagram

### Definition

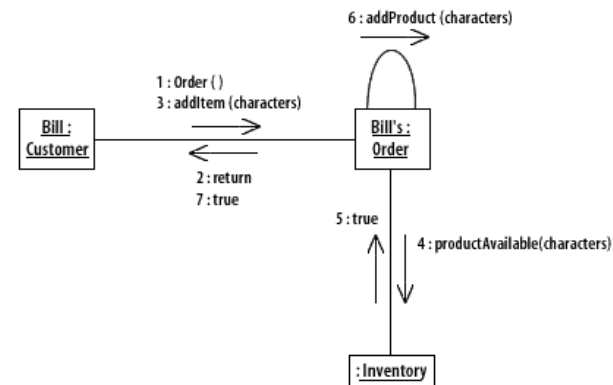
A **collaboration diagram** shows the interactions organized around the structure of a model, using either:

- a) classifiers (e.g. classes) and associations, or
- b) instances (e.g. objects) and links.

- 1) is an interaction diagram
- 2) is similar to the sequence diagram
- 3) reveals both structural and dynamic aspects of a collaboration
- 4) reveals the need for the associations in the class diagram

e-Macao-18-1-330

## Collaboration Diagram Example



e-Macao-18-1-331

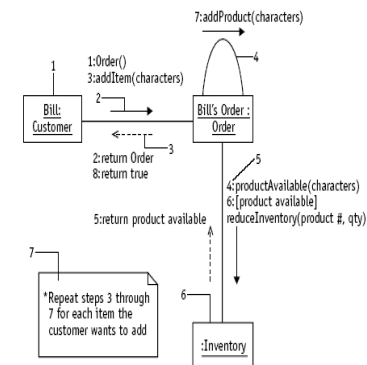
## Notation 1

- 1) A collaboration diagram shows a graph of either instances linked to each other or classifiers and associations.
- 2) Navigability is shown using arrow heads on the lines representing links.
- 3) An arrow next to a line indicates a stimuli or message flowing in the given direction.
- 4) The order of interaction is given with a number.

e-Macao-18-1-332

## Notation 2

1. object
2. synchronous event or procedure call
3. simple return
4. self-reference
5. sequence number
6. anonymous object
7. iteration comment



e-Macao-18-1-333

## Collaboration Objects

- 1) the backbone of the collaborating diagram
- 2) may have fully qualified name e.g. Bill of class Customer
- 3) may be anonymous
- 4) same as in the sequence diagram

e-Macao-18-1-334

## Collaboration Messages

- 1) involves sending of messages between class roles over associations
- 2) messages could be:
  - a) synchronous events: requires a reply
  - b) return: a reply message
  - c) asynchronous: does not require a reply
- 3) same as in sequence diagram

e-Macao-18-1-335

## Self Reference

- 1) a message from an object to itself
- 2) sender and receiver of the message is the same object
- 3) self invocation

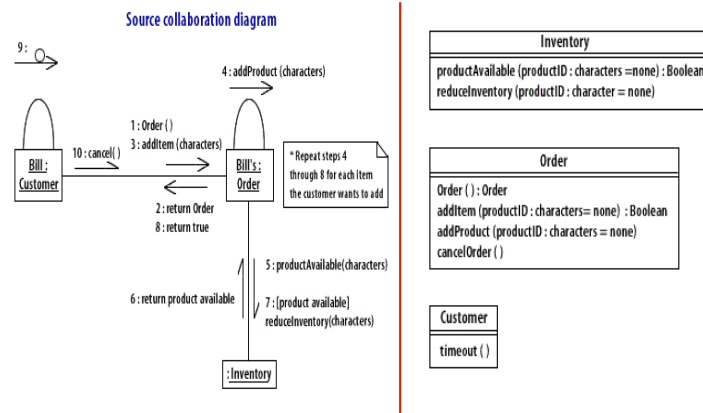
e-Macao-18-1-336

## Sequence Number

- 1) collaboration diagram does show explicitly passage of time
- 2) sequence numbers to show order of execution for the messages
- 3) no particular standard for numbering

e-Macao-18-1-337

## Collaboration and Object Diagram



e-Macao-18-1-338

## Building Collaboration Diagram

- 1) place participating objects on the diagram
- 2) draw the links between the objects using the class diagram as your guide
- 3) add each event by placing the message arrow parallel between the two objects
- 4) position the arrow to point from the sender to the receiver
- 5) number the messages in order of execution
- 6) repeat steps 3 and 4 until the entire scenario has been modeled

e-Macao-18-1-339

## Task 44

- 1) Consider the use case selected in Task 22. List an instance for each of the classes involved in the collaboration for this use case.
- 2) Provide a collaboration diagram for involving the objects listed in 1.

### A.5.3. Component Diagrams

## Architecture Modelling

### Component Diagrams

e-Macao-18-1-341

## Course Outline

1) Object Orientation

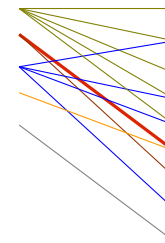
2) UML Basics

3) UML Modelling:

- a) Requirements
- b) Architecture
- c) Design
- d) Implementation
- e) Deployment

4) Unified Process

5) UML Tools



UML Diagrams:

- 1. use case
- 2. class
- 3. object
- 4. sequence
- 5. state
- 6. component
- 7. collaboration
- 8. activity
- 9. deployment

e-Macao-18-1-342

## Component

### Definition

A **component** is a physical, replaceable part that conforms to and provides the realization of a set of interfaces.

A component:

- 1) encapsulates the implementation of classifiers residing in it
- 2) does not have its own features, but serves as a mere container for its elements
- 3) are replaceable or substitutable parts of a system

e-Macao-18-1-343

## Component Example

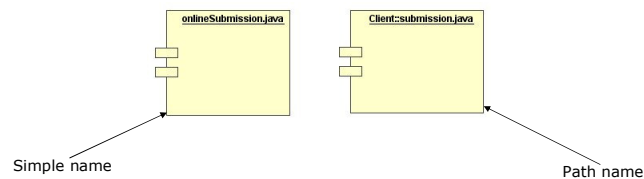


A component named OrderEntry.exe

e-Macao-18-1-344

## Component Names

- 1) component must have a unique name
- 2) name is a textual string which may be written as a simple name or with a path name.



e-Macao-18-1-345

## Component and Classes 1

### similarities

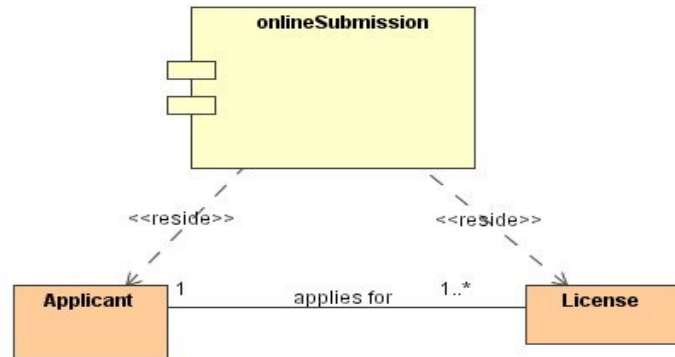
- 1) both may realize a set of interfaces
- 2) both may participate in dependencies, generalizations and associations
- 3) both may be nested
- 4) both may have instances
- 5) both may participate in an interaction

### differences

- 1) classes represent logical abstraction while components represent physical things
- 2) components represent the physical packaging of logical components and are at a different level of abstraction
- 3) classes may have attributes and operations whereas components only have operations reachable only through their interfaces

e-Macao-18-1-346

## Component and Classes 2



e-Macao-18-1-347

## Components and Interfaces

### Definition

An **interface** is a collection of operations that are used to specify a service of class or components.

Interfaces:

- 1) represent the major seam of the system
- 2) are realized by components in implementation
- 3) promotes the deployment of systems whose services are location independent and replaceable

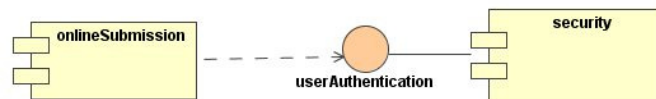
e-Macao-18-1-348

## Interface Notation 1

Relationship between a component and its interface may be shown in two ways:

### Style 1

- a) interface in elided iconic form
- b) component that realize the interface is connected to the interface using an elided realization relationship



e-Macao-18-1-349

## Interface Notation 2

### Style 2:

- a) interface is presented in an enlarged form, possibly revealing operations elided iconic form
- b) realizing component is connected to it using a full realization relationship





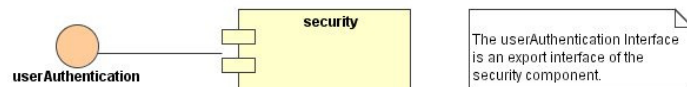
e-Macao-18-1-350

## Export Interface

### Definition

An interface that a component realizes is called an **export interface**, meaning an interface that the component provides as a service to other components.

Components may export more than one interface.



e-Macao-18-1-351

## Import Interface

### Definition

An interface that a component uses is called the **import interface**, meaning the interface that the component relies upon to implement its own behavior.

Components may import more than one interface.

They may also import and export interfaces at the same time.



e-Macao-18-1-352

## Component Replaceability

- 1) The key intent of any component-based system is to permit the assembly of systems from replaceable parts.
- 2) A system can be:
  - a) created out of existing components
  - b) evolved by adding new components and replacing old ones without rebuilding the system
- 3) Interfaces allow easy reconfiguration of component-based systems.

e-Macao-18-1-353

## Task 45

- 1) Identify at least three major components for the restaurant license application.
- 2) Describe the interface for each of the components.

e-Macao-18-1-354

## Types of Components

Three types: deployment, work product and execution.

- 1) **deployment**:  
component necessary and sufficient to form an executable system such as DLLs and EXEs
- 2) **work product component**:  
residue of development process consisting of things like source code files and data files from which deployment components are created
- 3) **execution component**:  
created as a consequence of an executing system

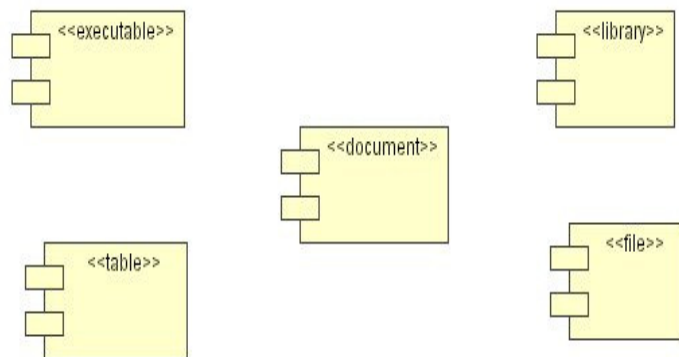
e-Macao-18-1-355

## Component Stereotypes 1

- 1) **executable**: specifies that a component may be executable on a node
- 2) **library**: specifies a static or dynamic object library
- 3) **table**: specifies a component that represents a database table
- 4) **file**: specifies a component that represents a document containing source code or data
- 5) **document**: specifies a component that represents a document

e-Macao-18-1-356

## Component Stereotypes 2



e-Macao-18-1-362

## Task 46

- 1) Add possible implementing artifacts for components listed in Task 45.
- 2) Using a UML tool, produce a component diagram containing the components in Task 45 and the implementing artifacts listed in question 1.

#### A.5.4. Packages

## Architecture Modelling

### Packages

e-Macao-18-1-364

## Packages

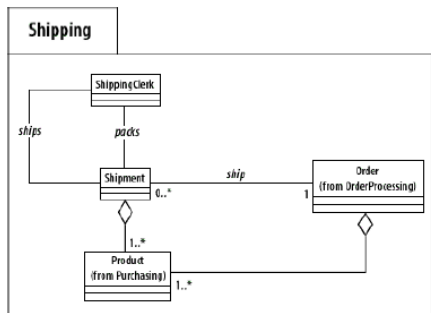
Packages:

- 1) are general purpose mechanism for organizing modelling elements into groups
- 2) group elements that are semantically close and that tend to change together

Packages should be loosely coupled, highly cohesive, with controlled access to its contents.

e-Macao-18-1-365

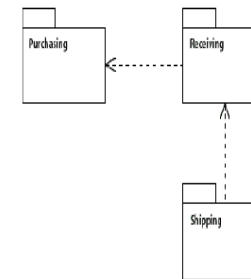
## Package Example 1



e-Macao-18-1-366

## Package Notation

- drawn as a tabbed folder
- packages references one another using standard dependency notation
- for instance: **Purchasing** package depends on **Receiving** package
- packages and their dependencies may be stereotyped



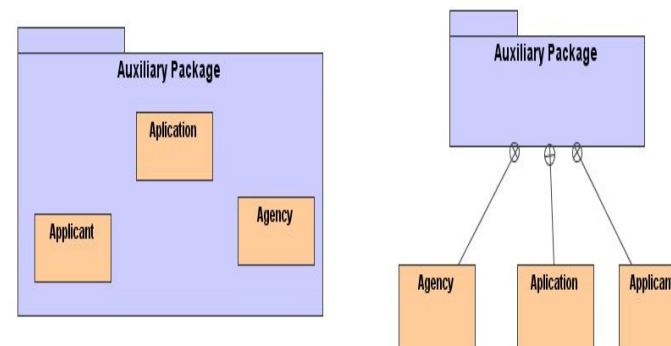
e-Macao-18-1-367

## Owned Elements

- 1) A package may own other elements for instance: classes, interfaces, components, nodes, collaborations, use cases, diagrams and other packages.
- 2) "Owning" is a **composite** relationship.
- 3) A package forms a namespace, thus elements of the same kind must be named uniquely. For example you cannot have two classes in the same package with the same name.

e-Macao-18-1-368

## Owned Elements Example



e-Macao-18-1-369

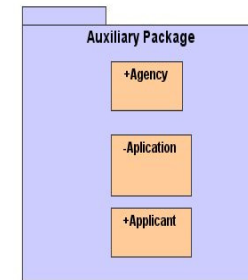
## Visibility

- 1) **visibility** of owned elements is similar to visibility of attributes and operations of classes
- 2) owned elements are visible to importing or enclosing package contents
- 3) protected elements can also be seen by children packages
- 4) private elements cannot be seen outside their owning package

e-Macao-18-1-370

## Visibility Example

- 1) "Auxiliary Package" owns:
  - a) Agency
  - b) Application
  - c) Applicant
- 2) package importing "Auxiliary Package" will have access to the "Agency" element but not "Application"



e-Macao-18-1-371

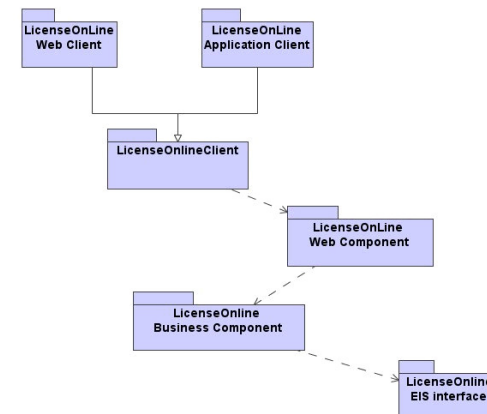
## Package Stereotypes

UML provides five stereotype that apply to packages:

- 1) **Façade**: a package that is only a view on some other packages
- 2) **Framework**: a package that consists mainly of patterns
- 3) **Stub**: specifies a package that serves as a proxy for the public contents of another package
- 4) **Subsystem**: a package representing an independent part of the entire system being modeled
- 5) **System**: specifies a package representing the entire system being modeled

e-Macao-18-1-372

## Package Example 1



e-Macao-18-1-373

## Task 47

---

- 1) Consider an application currently running in your organization. List the major subsystems of the application
- 2) Using a UML tool, provide an architectural model in terms of these subsystems using packages

### A.5.5. Frameworks and Patterns

## Architecture Modelling Patterns

e-Macao-18-1-375

## Course Outline

1) Object Orientation

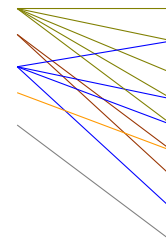
2) UML Basics

3) UML Modelling:

- a) Requirements
- b) Architecture
- c) Design
- d) Implementation
- e) Deployment

4) Unified Process

5) UML Tools



UML Diagrams:

- 1. use case
- 2. class
- 3. object
- 4. sequence
- 5. state
- 6. component
- 7. collaboration
- 8. activity
- 9. deployment

e-Macao-18-1-376

## Patterns – What are they?

“Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way the you can use this solution a million times over, without doing it the same way twice.”

– Christopher Alexander (Patterns in buildings and towns)

e-Macao-18-1-377

## Patterns Definition

### Definition

A generalized solution to a problem in a given context where each pattern has a description of the problem, solution context in which it applies, and heuristics including use advantages, disadvantages and trade-offs.

Patterns:

- identify, document, and classify best practices in OOD
- generalize the use and application of a society of elements

e-Macao-18-1-378

## Pattern Elements

- 1) **name**: a handle which describes the design problem, its solution, and consequences in a word or two
- 2) **problem**: describes when to apply the pattern and explains the problem and its context
- 3) **solution**: elements that must make up the design, their relationships, responsibilities and collaborations.
- 4) **consequences**: associated trade-offs in using the pattern.

e-Macao-18-1-379

## Pattern Example 1

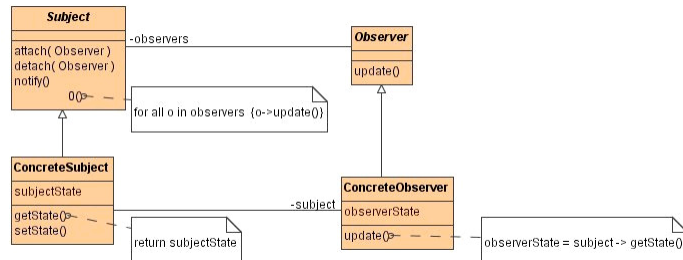
- 1) **Name**: Observer
- 2) **Problem**:
  - a) when an abstraction has two aspects, one dependent on the other
  - b) when a change to one object requires changing others, and you don't know how many objects need to be changed
  - c) when an object should be able to notify other object without making assumptions about who these objects are



e-Macao-18-1-380

## Pattern Example 2

### 3) Solution:



e-Macao-18-1-382

## Pattern Example 4

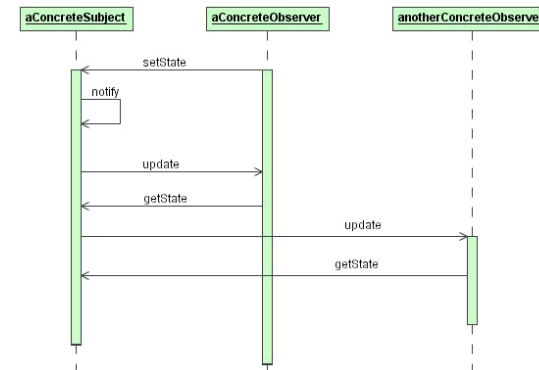
### 4) Consequences:

- a) abstract coupling between Subject and Observer
- b) support for broadcast communication
- c) unexpected updates

e-Macao-18-1-381

## Pattern Example 3

### 3) Solution:



e-Macao-18-1-383

## Types of Patterns

### 1) Creational Patterns

- a) instantiation of objects
- b) decoupling the type of objects from the process of constructing that object

### 2) Structural and Architectural patterns

- a) organization of a system
- b) larger structures composed from smaller structures.

### 3) Behavioural Patterns

- a) assigning responsibilities among a collection of objects.

e-Macao-18-1-384

## Framework 1

### Definition

A **Framework** is a reusable software architecture that provides the generic structure and behaviour for a family of software applications, along with a context that specifies their collaboration and use.

Framework:

- 1) collection of patterns defined as template collaborations with supporting elements.
- 2) skeletal solution in which specific element must be plugged in order to establish an actual solution.
- 3) depicted as packages stereotyped with the "framework" keyword

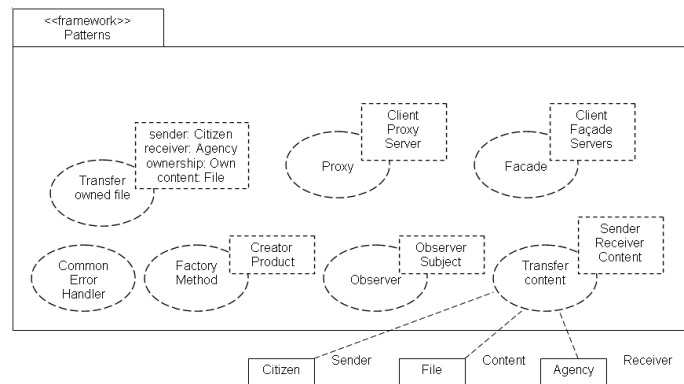
e-Macao-18-1-385

## Framework 2

- 1) made up of a set of related classes that can be specialized or instantiated to implement an application
- 2) lack the necessary application specific functionality and therefore not immediately useful
- 3) prefabricated structure or template of a working application in which "plug-points" or "hot spots" are not implemented or are given overridable implementations
- 4) patterns can be used to document frameworks
- 5) they are physical realization of one or more patterns

e-Macao-18-1-386

## Framework Example



e-Macao-18-1-387

## Architecture Patterns

- 1) Repository
- 2) Model-View-Controller (MVC)
- 3) Client-Server
- 4) Peer-to-Peer
- 5) Pipe-and-Filter

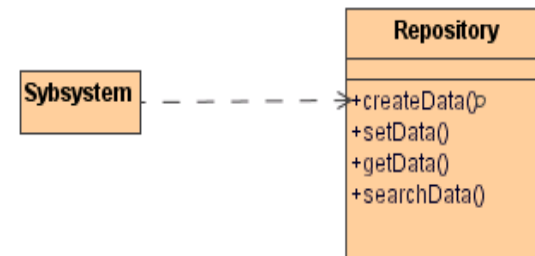
e-Macao-18-1-388

## Repository Architecture 1

- 1) subsystems access and modify data from a single data structure called the repository
- 2) subsystems are relatively independent and interact through the central data structure
- 3) control flow can be dictated either by the central repository or by the subsystem
- 4) it is usually employed in database applications, compilers and software development environment

e-Macao-18-1-389

## Repository Architecture 2



*UML Class Diagram Describing Repository Architecture*

e-Macao-18-1-390

## MVC Architecture 1

Subsystems are classified into three different types:

- a) **model** subsystems – maintains domain knowledge
- b) **view** subsystems – displays information to users
- c) **controller** subsystem – controls sequence of interaction

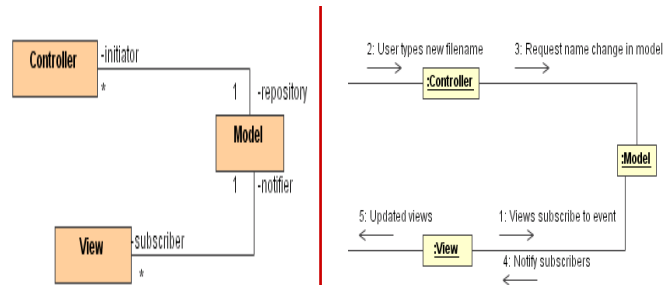
e-Macao-18-1-391

## MVC Architecture 2

- 1) model subsystems are written independently of view or controller subsystems
- 2) changes in model's state are propagated to the view subsystem through the subscribe/notify protocol
- 3) MVC architecture is a special case of repository architecture; model is the central repository and the controller subsystem dictates control flow

e-Macao-18-1-392

## MVC Architecture 3



*Structural and Behavioural Description of MVC Architecture*

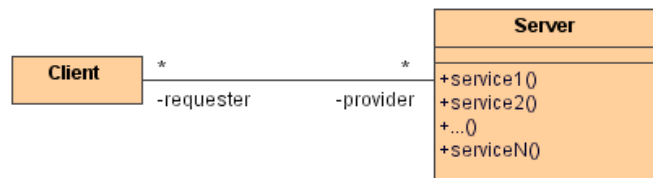
e-Macao-18-1-393

## Client – Server Architecture 1

- 1) two kinds of subsystems – the **Server** and **Client**
- 2) **server** subsystem provides services to instances of the other subsystems called **clients**; which interacts with the users
- 3) service requests are usually through remote procedure call or some other distributed programming techniques
- 4) control flow in clients and servers is independent except for synchronization to manage requests and receive results
- 5) there may be multiple servers subsystems like in the case of the world-wide web

e-Macao-18-1-394

## Client – Server Architecture 2



*Client – Server Architecture - Structural Description*

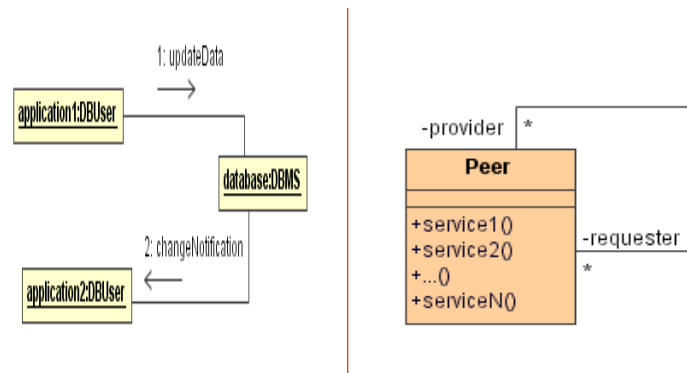
e-Macao-18-1-395

## Peer-to-Peer Architecture 1

- 1) generalization of the client-server architecture in which subsystems can be clients or servers dynamically
- 2) control flow within subsystems is independent from the others except for synchronization on requests
- 3) more difficult to design than client server systems as there are possibilities of deadlocks

e-Macao-18-1-396

## Peer-to-Peer Architecture 2



Peer-to-Peer Architecture – Behavioural and Structural Descriptions

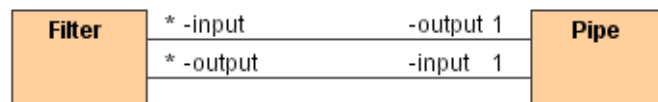
e-Macao-18-1-397

## Pipe-and-Filter Architecture 1

- 1) subsystems process data received from a set of input and send results to other subsystems via set of outputs
- 2) subsystems are called **filters** and the association between filters are called **pipes**
- 3) filters only know about the content and format of the data received on the input pipes and not the input filters
- 4) each filter is executed concurrently and synchronization is done via the pipes
- 5) pipes and filters can be reconfigured as required

e-Macao-18-1-398

## Pipe-and-Filter Architecture 2



e-Macao-18-1-399

## Task 48

1. The MVC architecture is better than the Client-Server architecture. Do you agree with this statement?
2. List two reasons for your agreement or disagreement with the statement in question 1.

e-Macao-18-1-400

## Summary 1

- 1) Architecture is concerned with the structural organization of system components as well as how they interact to provide the system's overall behaviour or functionality.
- 2) A collaboration is a society of classes which provides some cooperative behavior that is more than the sum of all its parts.
- 3) Collaborations have both structural and behavioral aspects.
- 4) Collaborations may realize a use case or an operation.

e-Macao-18-1-401

## Summary 2

- 5) A collaboration diagram shows interactions organized around the structure of a model, using either classes and associations or instances and links.
- 6) A component is a physical, replaceable part that conforms to and provides the realization of a set of interfaces.
- 7) An interface is a collection of operation that are used to specify a service of class or components.

e-Macao-18-1-402

## Summary 3

- 8) There are three categories of components: deployment, work product and execution.
- 9) Components may be stereotyped as executable, library, table, file or document.
- 10) Component diagram consists of specifying classes, implementing artifacts, components, interfaces and relationships between these model elements.

e-Macao-18-1-403

## Summary 4

- 11) Packages are general purpose mechanisms for organizing modeling elements into groups.
- 12) Well structured packages must be loosely coupled and very cohesive.
- 13) Basic architectural patterns or styles include: repository, MVC and client-server.

## A.6. Design

### A.6.1. Software Design

# Design Modelling

e-Macao-18-1-405

## Course Outline

1) Object Orientation

2) UML Basics

3) UML Modelling:

a) Requirements

b) Architecture

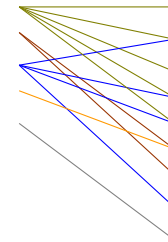
c) Design

d) Implementation

e) Deployment

4) Unified Process

5) UML Tools



UML Diagrams:

1. use case
2. class
3. object
4. sequence
5. state
6. component
7. collaboration
8. activity
9. deployment

e-Macao-18-1-406

## System Design

“There are two ways of constructing a software design: one way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies”

C.A.R. Hoare

e-Macao-18-1-407

## What is System Design

### Definition

**System Design** is the transformation of analysis models of the problem space into design models (based on the solution space).

It involves selecting strategies for building the system e.g. software/hardware platform on which the system will run and the persistent data strategy.

e-Macao-18-1-408

## System Design – Overview 1

### Analysis produces:

- a) a set of non-functional requirements and constraints
- b) a use case model describing functional requirements
- c) a conceptual model, describing entities
- d) a sequence diagram for each use case showing the sequence of interaction among objects
- e) statechart diagrams showing possible states of objects

e-Macao-18-1-409

## System Design – Overview 2

### Design results in:

- a) a list of design goals – qualities of the system
- b) software architecture describing:
  - a) the subsystem responsibilities,
  - b) dependencies among subsystems,
  - c) subsystem mapping to hardware,
  - d) major policy decision such as control flow, access control, and data storage



e-Macao-18-1-410

## System Design – Activities

- 1) definition of design goals
- 2) decomposition of system into sub-system
- 3) selection of off-the-shelf and legacy components
- 4) mapping of sub-systems to hardware
- 5) selection of persistent data management infrastructure
- 6) selection of access control policy
- 7) selection of a global control flow mechanism
- 8) handling of boundary conditions

e-Macao-18-1-411

## Some Design Activities 1

- 1) **Define goals**
  - a) derived from non-functional requirements
- 2) **Hardware and software mapping**
  - a) what computers (nodes) will be used?
  - b) which node does what?
  - c) how do nodes communicate?
  - d) what existing software will be used and how?
- 3) **Data management**
  - a) which data needs to be persistent?
  - b) where should persistent data be stored?
  - c) how will the data be stored?

e-Macao-18-1-412

## Some Design Activities 2

- 4) **Access control**
  - a) who can access which data?
  - b) can access control be changed within the system?
  - c) how is access control specified and realized?
- 5) **Control flow**
  - a) how does the system sequence operations?
  - b) is the system event-driven?
  - c) does it need to handle more than one user interaction at a time?

e-Macao-18-1-413

## What is Object Design? 1

- 1) system design describes the system in terms of its architecture:
  - a) subsystem decomposition,
  - b) global control flow and
  - c) persistency management
- 2) object design includes:
  - a) service specification
  - b) component selection
  - c) object model restructuring
  - d) object model optimization

e-Macao-18-1-414

## What is Object Design? 2

- 1) **service specification** – precise description of class interface
- 2) **component selection** – identification of off-the-shelf components and solution objects
- 3) **object model restructuring** – transform the object design model to improve understandability and extensibility
- 4) **object model optimization** – transform object model to address performance (response time, memory etc.)

e-Macao-18-1-415

## Object Design Activities 1

- |   |  |
|---|--|
| <ol style="list-style-type: none"><li>1) <b>specification</b><ol style="list-style-type: none"><li>a) missing attributes and operations</li><li>b) type signatures and visibility</li><li>c) constraints</li><li>d) exceptions</li></ol></li><li>2) <b>component selection</b><ol style="list-style-type: none"><li>a) adjusting class libs</li><li>b) adjusting application frameworks</li></ol></li></ol> | <ol style="list-style-type: none"><li>3. <b>restructuring</b><ol style="list-style-type: none"><li>a) realizing associations</li><li>b) increasing reuse</li><li>c) removing implementation dependencies</li></ol></li><li>4. <b>optimization</b><ol style="list-style-type: none"><li>a) access paths</li><li>b) collapsing objects</li><li>c) caching results of expensive comp</li><li>d) delaying expensive comp</li></ol></li></ol> |
|---|--|

## A.6.2. Design Class Diagrams

# Design Modelling

## Class Diagrams

e-Macao-18-1-417

## Course Outline

1) Object Orientation

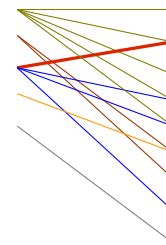
2) UML Basics

3) UML Modelling:

- a) Requirements
- b) Architecture
- c) Design
- d) Implementation
- e) Deployment

4) Unified Process

5) UML Tools



UML Diagrams:

- 1. use case
- 2. class
- 3. object
- 4. sequence
- 5. state
- 6. component
- 7. collaboration
- 8. activity
- 9. deployment

e-Macao-18-1-418

## Design Classes

### Class:

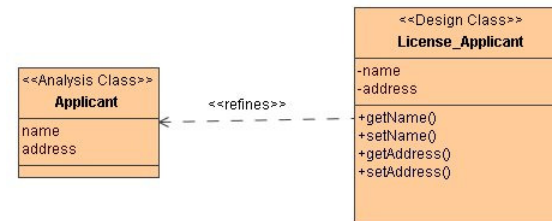
- basis for other diagrams and UML models
- describes the static view of the system

### Design Classes:

- different from domain classes
- design classes express the definitions of classes as software components

e-Macao-18-1-419

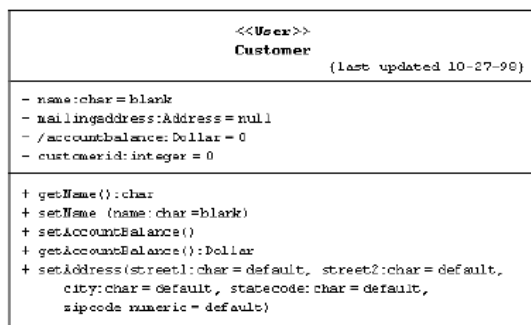
## Design Class Example 1



e-Macao-18-1-420

## Design Class Example 2

Complete class with all three compartments



e-Macao-18-1-421

## Design Class Diagram

- 1) provides the specification for software classes and interfaces in an application
- 2) includes:
  - a) classes, associations and attributes
  - b) interfaces, with their operations and constants
  - c) methods
  - d) attribute type information
  - e) navigability
  - f) dependencies

e-Macao-18-1-422

## Creating Design Class Diag. 1

- 1) identify all classes participating in object interaction by analyzing the collaborations
- 2) present them in a class diagram
- 3) copy attributes from the associated concepts in the conceptual model
- 4) add methods names by analyzing the interaction diagrams
- 5) add type information to the attributes and methods

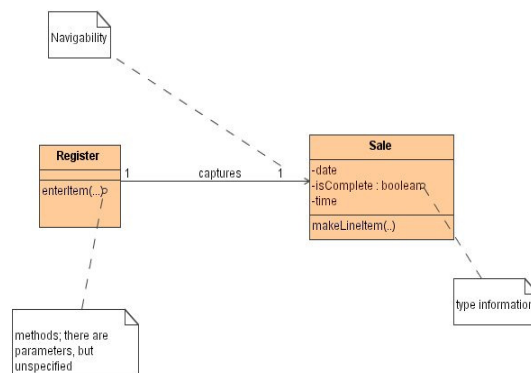
e-Macao-18-1-423

## Creating Design Class Diag. 2

- 6) add the association necessary to support the required attribute visibility
- 7) add navigability arrows necessary to the associations to indicate the direction of the attribute visibility
- 8) add dependency relationship lines to indicate non-attribute visibility

e-Macao-18-1-424

## Design Class Diagram Example



e-Macao-18-1-425

## Adding Classes

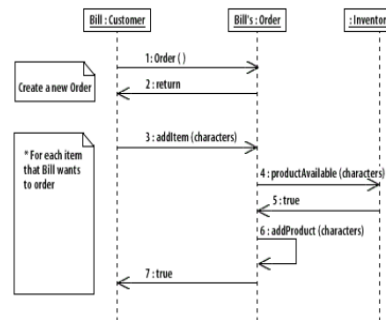
- 1) identify those classes that participate in the software solution
- 2) classes can be found by scanning all the interaction diagrams and listing the participating classes

e-Macao-18-1-426

## Adding Classes Example

Candidate classes:

- a) Customer
- b) Order
- c) Inventory



e-Macao-18-1-427

## Adding Attributes

- 1) after defining the classes, attributes must be added
- 2) attributes come from software requirements artifacts for example conceptual classes provide a substantial part of the attribute applicable to design classes

e-Macao-18-1-428

## Attribute Syntax

6 7 1 2 3 4 5  
 + ? / AttributeName:Data Type = Default Value {Constraints}

1. Derived attribute indicator: Optional
2. Attribute name: Required
3. Data type: Required
4. Assignment operator and default value: Optional
5. Constraints: Optional
6. Visibility: Required before code generation
7. Class attribute: Optional

e-Macao-18-1-429

## Adding Operations

The operations of each class can be identified by analyzing the interaction diagrams:

- a) messages will include calls to other objects
- b) self references

e-Macao-18-1-430

## Operations Syntax

```

4 5      1      6      7      2
+ $ OperationName (ArgName: Data Type,...):
  Return Data Type (Constraints)
      3          8
  
```

1. Operation name: Required
2. Any number of arguments is allowed
3. Return data type: Required for a return value, but return values themselves are optional
4. Visibility: Required before code generation
5. Class operation: Optional
6. Argument name: Required for each argument, but arguments themselves are optional
7. Argument data type: Required for each argument, but arguments themselves are optional
8. Constraints: Optional

e-Macao-18-1-431

## Adding Visibility

### Visibility:

- 1) scope of access allowed to a member of a class
- 2) applies to attributes and operations

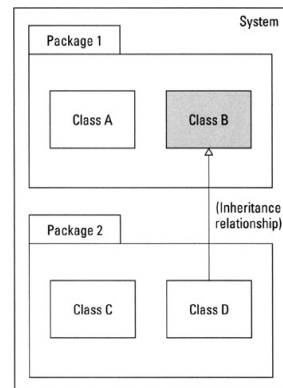
### UML visibility maps to OO visibilities:

- 1) **private scope**: within a class (-)
- 2) **package scope**: within a package (~)
- 3) **public scope**: within a system (+)
- 4) **protected scope**: within an inheritance tree (#)

e-Macao-18-1-432

## Private Visibility

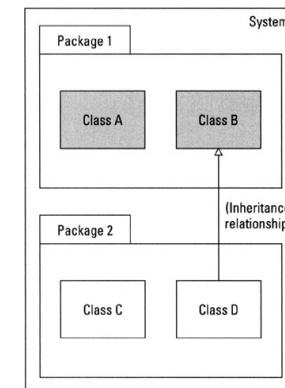
- 1) private element is only visible inside the namespace that owns it
- 2) notation is "-"
- 3) useful in encapsulation.



e-Macao-18-1-433

## Package Visibility

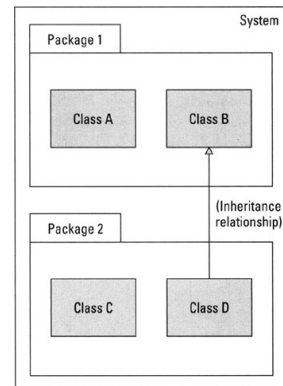
- 1) package element is owned by a namespace that is not a package, and is visible to elements that are in the same package as its owning namespace
- 2) notation is "~"



e-Macao-18-1-434

## Public Visibility

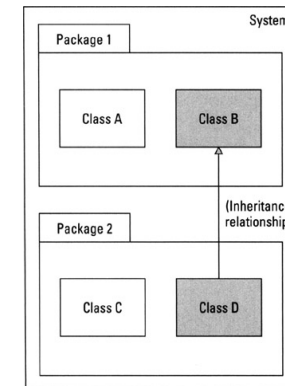
- 1) public element is visible to all elements that can access the contents of the namespace that owns it
- 2) notation is "+"



e-Macao-18-1-435

## Protected Visibility

- 1) a protected element is visible to elements that are in the generalization relationship to the namespace that owns it
- 2) notation is "#"



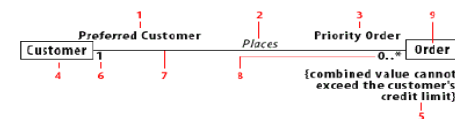
e-Macao-18-1-436

## Adding Associations

- 1) end of an association is called a role
- 2) roles may be decorated with a navigability arrow
- 3) navigability is a property of a role that indicates that it is possible to navigate uni-directionally across the association from objects of the source to target
- 4) navigability indicates attributes visibility
- 5) implementation assumes the source class has an attribute that refers to an instance of the target class

e-Macao-18-1-437

## Association Syntax



1. Role: Must have a name or roles or both
2. Association name: Must have a name or roles or both
3. Role: Must have a name or roles or both
4. Class
5. Constraint: Optional
6. Multiplicity: Required
7. Association
8. Multiplicity: Required
9. Class



e-Macao-18-1-438

## Adding Dependency

- 1) useful to depict non-attribute visibility between classes
- 2) useful when there are parameters, global or locally declared visibility

e-Macao-18-1-439

## Task 49

Using a UML tool, produce a design class diagram based on the conceptual classes and dynamic models already developed in previous tasks:

- 1) add attributes if needed
- 2) add operations
- 3) add associations with names, roles and multiplicity
- 4) add necessary dependencies

### A.6.3. Activity Diagrams

## Design Modelling

### Activity Diagrams

e-Macao-18-1-441

## Course Outline

1) Object Orientation

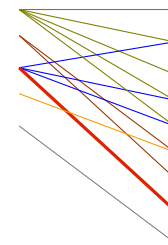
2) UML Basics

3) UML Modelling:

- a) Requirements
- b) Architecture
- c) Design
- d) Implementation
- e) Deployment

4) Unified Process

5) UML Tools



UML Diagrams:

- 1. use case
- 2. class
- 3. object
- 4. sequence
- 5. state
- 6. component
- 7. collaboration
- 8. activity
- 9. deployment

e-Macao-18-1-442

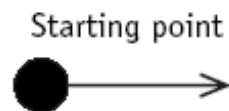
## Activity Diagrams

- 1) variation of the state machine in which the states represent the performance of actions or sub-activities
- 2) transitions are triggered by the completion of the actions or sub-activities
- 3) one of the five diagrams in UML for modeling the dynamic aspect of a system.  
Others: **sequence**, **collaboration**, **statechart**, **use cases**
- 4) attached through a model to use cases, or to the implementation an operations
- 5) focuses on flows driven by internal processing (as opposed to external events)

e-Macao-18-1-443

## Start State

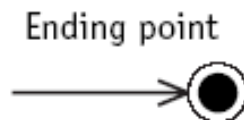
- 1) signals the beginning of the activity diagram
- 2) indicated as a solid dot



e-Macao-18-1-444

## End State

- 1) signals the end of the activity diagram
- 2) indicated as a bull eye



e-Macao-18-1-445

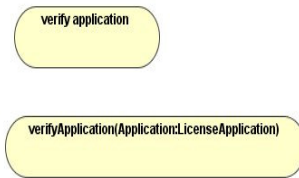
## Action State

- 1) shorthand for a state with an entry action and at least one outgoing transition involving the implicit event of completing the entry action
- 2) there may be several outgoing transitions with guard conditions
- 3) models a step in the execution of a workflow

e-Macao-18-1-446

## Action State – Notation

- 1) a shape with straight top and bottom and with convex arcs on the two sides
- 2) action expression is placed in the action state symbol
- 3) action expression may be described as natural language or pseudo code



e-Macao-18-1-447

## Subactivity State

- 1) it invokes an activity graph
- 2) executes the associated activity graph
- 3) completes the subactivity graph is not exited until the final state of the nested graph is reached
- 4) a single activity graph may be invoked by many subactivity states

e-Macao-18-1-448

## Subactivity State – Notation

- 1) shown the same way as an action state with the addition of an icon in the lower right corner depicting a nested activity diagram
- 2) subactivity name is placed in the symbol



e-Macao-18-1-449

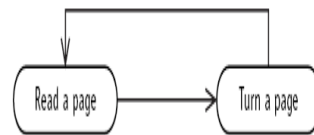
## Transitions 1

- 1) specify the flow of control from one action or activity state to another
- 2) trigger-less
- 3) passes control immediately on completion of the work in the source state to the next activity state
- 4) causes the state's exit action (if any) to be executed

e-Macao-18-1-450

## Transitions 2

- 1) after reading a page, the page is turned
- 2) this is repeated continuously



e-Macao-18-1-451

## Decisions

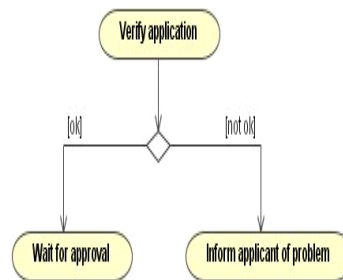
- 1) Decisions are expressed as guard conditions.
- 2) Different guards are used to indicate different possible transitions that depend on boolean conditions of the owning objects.
- 3) The predefined guard else may be defined for at most one outgoing transition which is enabled if all the guards labeling the other transitions are false.

e-Macao-18-1-452

## Decision - Notation

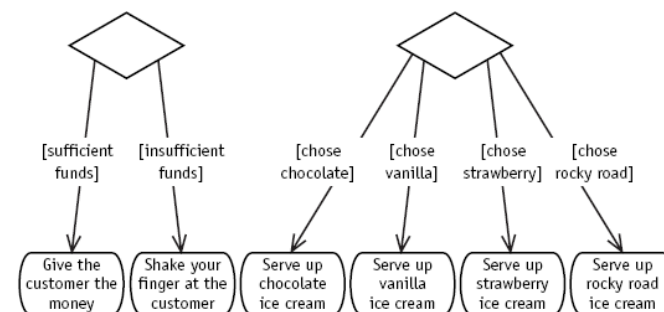
Decision may be shown by labeling multiple output transitions of an action with different guard conditions.

The icon provided for a decision is the traditional diamond shape, with one incoming arrow and with two or more outgoing arrows, each with a distinct guard condition with no event trigger.



e-Macao-18-1-453

## Decision Example



e-Macao-18-1-454

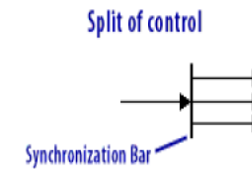
## Forks

- 1) used for modelling concurrency and synchronization in business processes
- 2) uses a synchronization bar
- 3) represents the splitting of a single flow of control into two or more concurrent flows of controls
- 4) indicates that activities of each of the these flows are truly concurrent when deployed across multiple nodes or sequentially interleaved if deployed on a single node

e-Macao-18-1-455

## Fork Example

A single control splitting to three concurrent flows



e-Macao-18-1-456

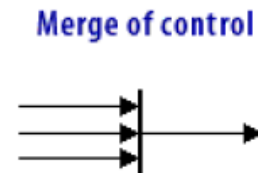
## Joins

- 1) represents the synchronization of two or more concurrent flows of control.
- 2) may have two or more incoming transitions and one outgoing transition
- 3) synchronizes concurrent flows, constraining each flow to wait for other incoming flows

e-Macao-18-1-457

## Join Example

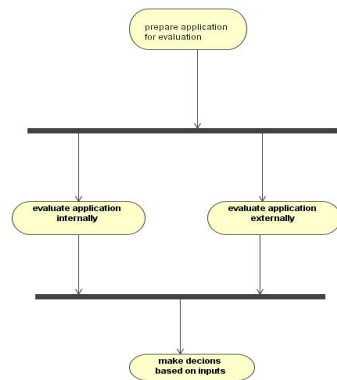
Three incoming flows synchronized and proceed as just one flow



e-Macao-18-1-458

## Forks and Join Example

- 1) application is prepared for evaluation
- 2) internal and external evaluation proceeds concurrently
- 3) decision is made only when the internal and external evaluation are completed



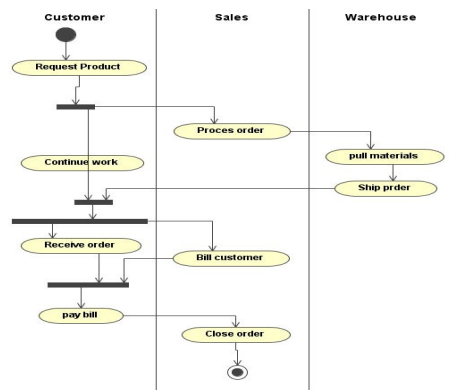
e-Macao-18-1-459

## Swimlanes

- 1) partitions the activity states into groups
- 2) represent the entity within the organization responsible for those activities
- 3) has a unique name within a diagram

e-Macao-18-1-460

## Swimlanes Example



e-Macao-18-1-461

## Object Flow

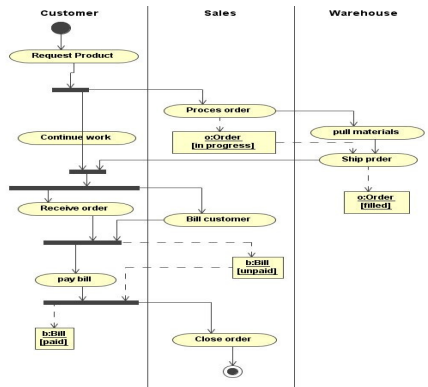
Objects maybe involved in the flow of control associated with an activity diagram.

For example in the workflow of processing an order as shown in the last example, we may wish to show how the state of the *order* and *Bill* objects change.

We can use dependency relationship to show how objects are created, modified and destroyed by transitions in the activity diagram

e-Macao-18-1-462

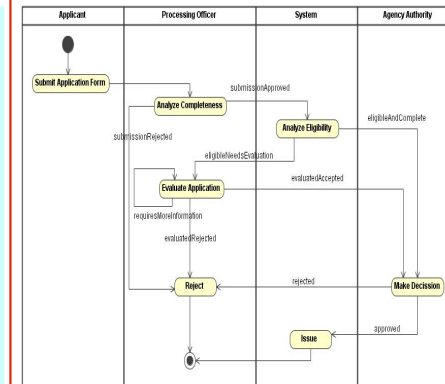
## Object Flow Example



e-Macao-18-1-463

## Task 50

- 1) Provide a narrative for the activity diagram shown.
- 2) Assume the procedure shown is a workflow for restaurant licensing, suggest possible changes to the model.
- 3) Show how the status of the application object changes along the flow.





#### A.6.4. Sequence Diagrams

## Design Modelling

### Sequence Diagrams

e-Macao-18-1-465

## Course Outline

1) Object Orientation

2) UML Basics

3) UML Modelling:

a) Requirements

b) Architecture

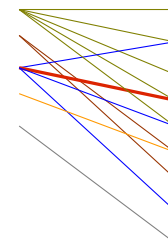
c) Design

d) Implementation

e) Deployment

4) Unified Process

5) UML Tools



UML Diagrams:

1. use case

2. class

3. object

4. sequence

5. state

6. component

7. collaboration

8. activity

9. deployment

e-Macao-18-1-466

## Design Sequence Diagrams

- 1) present the interactions between objects needed to provide a specific behavior
- 2) capture the sequence of events between participating objects
- 3) take into account temporal ordering
- 4) optionally shows which objects active at any time

e-Macao-18-1-467

## Design Sequence Diagrams

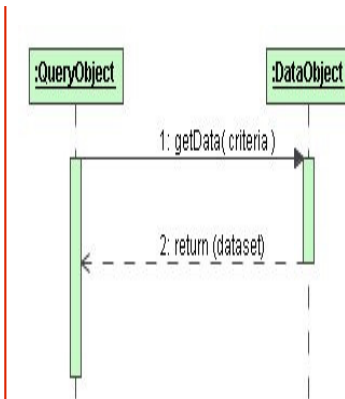
Design considerations:

- a) type of message
- b) timing of messages
- c) recursive messages
- d) object creation and destruction

e-Macao-18-1-468

## Synchronous Messages

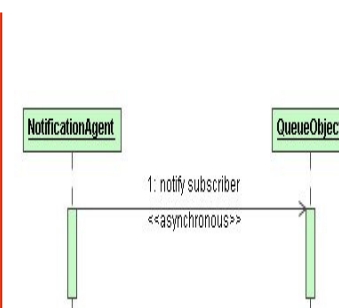
- 1) assumes that a return is needed
- 2) sender waits for the return before proceeding with any other activity
- 3) represented as a full arrow head
- 4) return messages are dashed arrows



e-Macao-18-1-469

## Asynchronous Messages

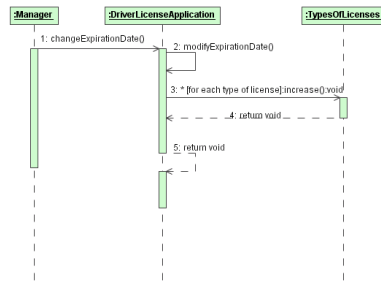
- 1) does not wait for a return message
- 2) exemplified by signals
- 3) sender only responsible for getting the message to the receiver
- 4) usually modeled using a solid line and a half arrowhead to distinguish it from the full arrowhead of the synchronous message



e-Macao-18-1-470

## Self-Reference Message

A **self-reference message** is a message where the sender and receiver are one and the same object.



- 1) in a self-reference message the object refers to itself when it makes the call
- 2) message 2 is only the invocation of some procedure that should be executed

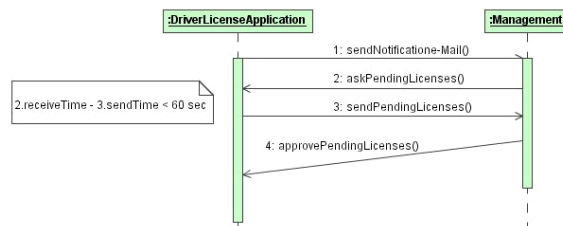
e-Macao-18-1-471

## Timed Messages

- 1) messages may have user-defined time attributes, such as **sentTime** or **receivedTime**
- 2) user-defined time attributes must be associated with message numbers
- 3) instantaneous messages are modeled with horizontal arrows
- 4) messages requiring a significant amount of time, it is possible to slant the arrow from the tail down to the head

e-Macao-18-1-472

## Timed Messages Example



For messages 1, 2 and 3 the time required for their execution is considered equal to zero.

Message 4 requires more time (time > 0) for its execution.

e-Macao-18-1-473

## Activation and Deactivation

Sequence diagrams can show object activation and deactivation.

**Activation** means that an object is occupied performing a task.

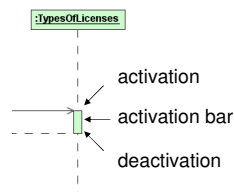
**Deactivation** means that the object is idle, waiting for a message.

e-Macao-18-1-474

## Activation and Focus of Control

Activation is shown by widening the vertical object lifeline to a narrow rectangle, called an **activation bar** or **focus of control**.

An object becomes active at the top of the rectangle and is deactivated when control reaches the bottom of the rectangle.



e-Macao-18-1-476

## Creation and Destruction

### Object Creation:

- if the object is created during the sequence execution it should appear somewhere below the top of the diagram.

### Object Destruction:

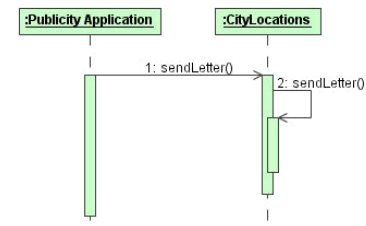
- if the object is deleted during the sequence execution, place an X at the point in the object lifeline when the termination occurs.

e-Macao-18-1-475

## Recursion

An object might also need to call a message recursively, this means to call the same message from within the message.

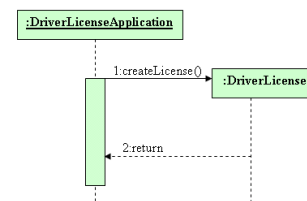
- 1) suppose that `cityLocations` is defined in the class diagram as a set of one or more apartments or houses
- 2) A letter could be sent to all apartments in a location as shown



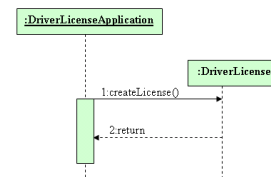
e-Macao-18-1-477

## Object Creation Example

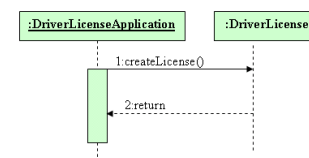
### Alternative A



### Alternative B

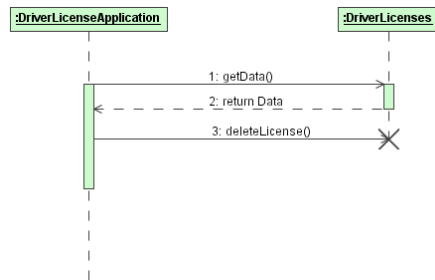


### Alternative C



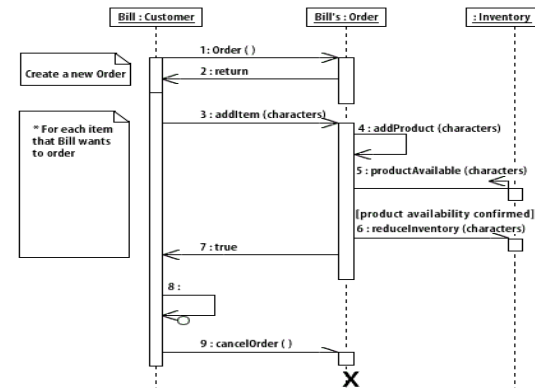
e-Macao-18-1-478

## Object Destruction Example



e-Macao-18-1-479

## Sequence Diagram Example



e-Macao-18-1-480

## Task 51

Consider the “submit new application” use case for restaurant license application.

Based on your design classes, provide a sequence diagram to show the interaction required for this use case.

### A.6.5. Statechart Diagrams

## Design Modelling

### State Diagrams

e-Macao-18-1-482

## Course Outline

1) Object Orientation

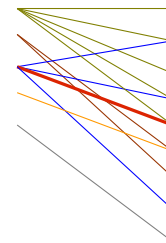
2) UML Basics

3) UML Modelling:

- a) Requirements
- b) Architecture
- c) Design
- d) Implementation
- e) Deployment

4) Unified Process

5) UML Tools



UML Diagrams:

- 1. use case
- 2. class
- 3. object
- 4. sequence
- 5. state
- 6. component
- 7. collaboration
- 8. activity
- 9. deployment

e-Macao-18-1-483

## Design Statechart Diagrams

Specifies detailed state information:

- 1) composite states
- 2) dynamic and static branching

e-Macao-18-1-484

## Design Statechart Diagrams

- 1) statechart diagrams illustrate how these objects behave internally
- 2) statechart diagrams relate events to state transitions and states
- 3) transitions change the state of the system and are triggered by events
- 4) design statechart diagrams provide detailed information on internal behaviors of objects

e-Macao-18-1-485

## Static Branch Point

Another pseudo state provided by UML is the **static branch point** that allows a transition to split into two or more paths.

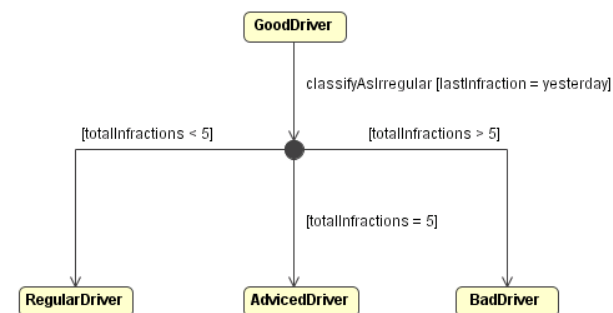
Notation:



It provides a means to simplify compound guard conditions by combining the like portions into a single transition segment, then branching based on the portions of the guard that are unique.

e-Macao-18-1-486

## Static Branch Point Example

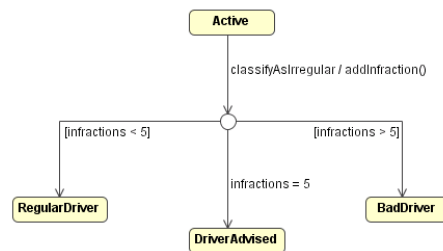


e-Macao-18-1-487

## Dynamic Branch Point

In other transitions, the destination is not known until the associated action has been completed.

Notation: ○



e-Macao-18-1-488

## Modelling Composite States

A **composite state** is simply a state that contains one or more statecharts diagrams.

Composite states may contain either:

- 1) a set of mutually exclusive states: literally like embedding a statechart diagram inside a state
- 2) a set of concurrent states: different states divided into regions, active at the same time

A composite state is also called a **super-state**, a generalized state that contains a set of specialized states called **sub-states**.

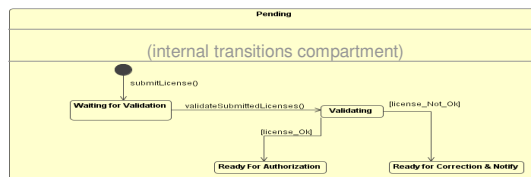
e-Macao-18-1-489

## Sub-States

A composite state (super-state) may be decomposed into two or more lower-level states (sub-states).

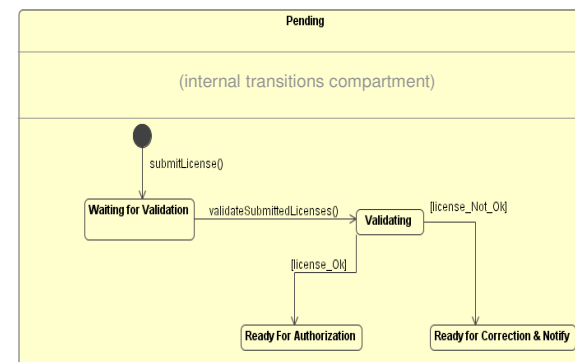
All the rules and notation are the same for the contained substates as for any statechart diagram.

Decomposition may have as many levels as needed.



e-Macao-18-1-490

## Sub-States Example



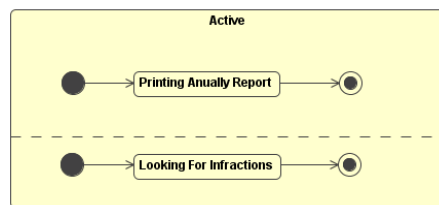


e-Macao-18-1-491

## Concurrent Sub-States

Modelling concurrent substates implies that you have many things occurring at the same time.

To isolate them, the composite state is divided into regions, and each region contains a distinct statechart diagram.



e-Macao-18-1-492

## Sub-Machine States

A **sub-machine state** is a shorthand for referring to an existing statechart diagram.

Within a composite state, it is possible to reference to a sub-machine state in the same way that a class may call a subroutine or a function of another class.

The composite state containing the sub-machine is called the **containing state machine**, and the sub-machine is called the **referenced state machine**.

Access to sub-machines states is through entry and exit points that are specified by **stub states**.

e-Macao-18-1-493

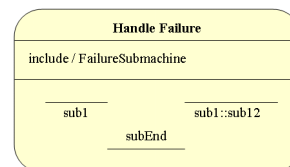
## Notation for Sub-Machine

The containing state icon models the reference to the sub-machine adding the keyword **include** followed by a slash plus the name of the submachine state.

The **stub state** uses a line with the state name placed near the line.

**Entry points** are represented with a line and the name under the line.

**Exit points** are represented with a line and the name over the line.



e-Macao-18-1-494

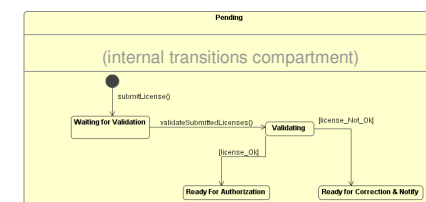
## Transitions to Sub-States 1

Alternative A:

- draw a transition pointing to the edge of the composite state icon
- it means that the composite state starts in the default initial state
- the default initial state is the sub-state associated with the initial state icon in the contained statechart diagram



The initial sub-state is **Waiting for Validation**



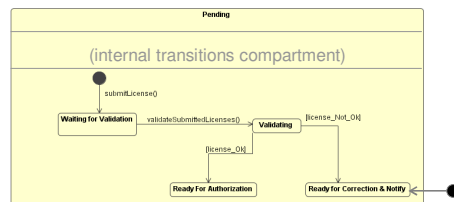
e-Macao-18-1-495

## Transitions to Sub-States 2

Alternative B:

- draw a transition through the edge of the super-state to the edge of the specific sub-state
- it means that the composite state starts in that specific sub-state

The initial sub-state is **Ready for Correction & Notify**



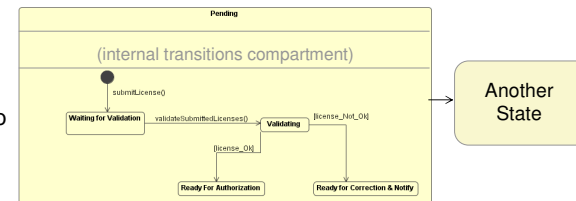
e-Macao-18-1-496

## Transitions from Sub-States 1

Alternative A:

- an event can cause the object to leave the super-state regardless of the current sub-state
- draw the transition from the edge of the super-state to the new state.

At any sub-state the object can do a transition to **AnotherState**

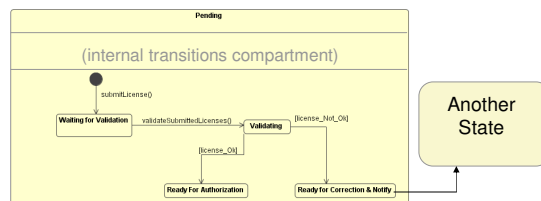


e-Macao-18-1-497

## Transitions from Sub-States 2

Alternative B:

- an event can cause the object to leave the super-state directly from a specific sub-state
- implies that the exit event may only happen when the object is in the specific sub-state
- draw the transition from the edge of the specific sub-state through the edge of the super-state.

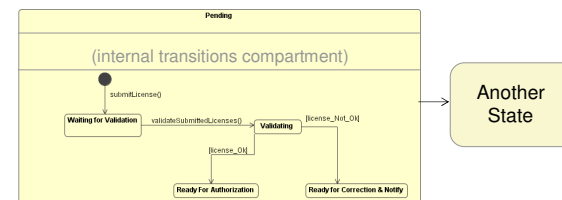


e-Macao-18-1-498

## Transitions from Sub-States 3

Alternative C:

- an object may leave a super-state because all the activities in the state and its sub-state has been completed
- it is called an **automatic transition**
- draw the transition from the edge of the super-state to the new state



e-Macao-18-1-499

## History Indicator 1

The history indicator is used to represent the ability for doing backtracking to a previous composite state.

Represents a pseudo-state and is a shorthand notation to solve a complex modelling problem.

May refer to:

- 1) **shallow history**: the object should return to the last sub-state on the top most layer of sub-states
- 2) **deep history**: the object needs to return to the exact sub-state from it which left, no matter how many layers down that is

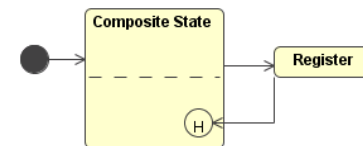
e-Macao-18-1-500

## History Indicator 2

Notation:

1) shallow history (H)

2) deep history (H\*)



e-Macao-18-1-501

## Split of Control

**Split of control** means that based on a single transition it is necessary to proceed with several tasks concurrently.

Notation:

- 1) a single transition divided into multiple arrows, each pointing to a different sub-state
- 2) the division is accomplished by the synchronization bar



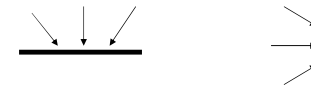
e-Macao-18-1-502

## Merge of Control

**Merge of control** means that based on the completion of a number of transitions it is necessary to proceed with a single task.

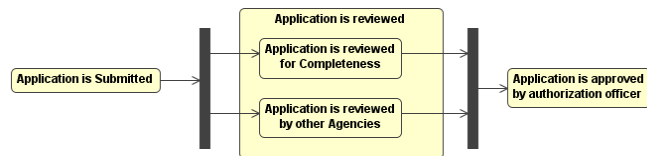
Notation:

- multiple transitions converge to a synchronization bar and only one transition outputs from the bar



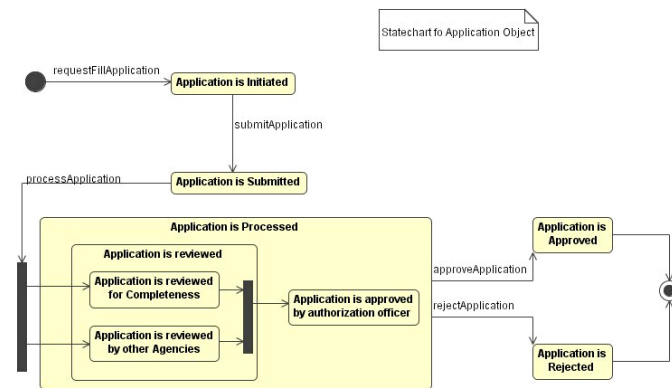
e-Macao-18-1-503

## Split/Merge Control Example



e-Macao-18-1-504

## Statechart Diagram Example



e-Macao-18-1-505

## Task 52

Consider the restaurant license application, provide a detailed statechart diagram for the "Application" object.

## A.6.6. Design Patterns

# Design Modelling Patterns

e-Macao-18-1-507

## Course Outline

1) Object Orientation

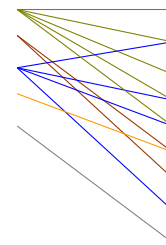
2) UML Basics

3) UML Modelling:

- a) Requirements
- b) Architecture
- c) Design
- d) Implementation
- e) Deployment

4) Unified Process

5) UML Tools



UML Diagrams:

- 1. use case
- 2. class
- 3. object
- 4. sequence
- 5. state
- 6. component
- 7. collaboration
- 8. activity
- 9. deployment

e-Macao-18-1-508

## Design Patterns 1

### Definition

Design patterns are partial solutions to common problems. They name, abstract, and identify the key aspects of common design structure that make it useful for creating reusable object-oriented design.

### Some common problems:

- a) separating interfaces from a number of alternate implementations
- b) wrapping around a set of legacy classes
- c) protecting a caller from changes associated with specific platforms

e-Macao-18-1-509

## Design Patterns 2

- 1) They are related to coding idioms that exist for programming languages.
- 2) Design patterns capture expert knowledge and design tradeoffs and support the sharing of architectural knowledge among developers.
- 3) Design patterns as a shared vocabulary can clearly document the software architecture of a system.
- 4) Allow design engineers relate to one another a a higher level of abstraction.

e-Macao-18-1-510

## Design Patterns: Description 1

Attribute	Description
Pattern Name and Classification	Conveys the essence of the pattern succinctly.
Intent	A short statement that answers the following questions: what does the design do? Its rationale and intent.
Also Known As	Other well known names for the patterns, if any
Motivation	A scenario that illustrates a design problem and how the class and object structures in the pattern solve the problem.
Applicability	What are the situations in which the design pattern can be applied?
Structure	Class diagram and sequence diagram for the classes involved in the pattern.
Participants	The classes and/or the object participating in the design pattern and their responsibilities.
Collaborations	How participants collaborate to carry out their responsibilities.

e-Macao-18-1-511

## Design Patterns: Description 2

Attribute	Description
Consequences	How does pattern support its objectives? What are the tradeoffs?
Implementation	What pitfalls, hints, or techniques you should be aware of when implementing the pattern
Sample Code	Code fragment to show implementation
Known uses	Example of patterns found in real systems
Related Patterns	What design patterns are closely related to this one?

e-Macao-18-1-512

## Design Patterns Catalog

1. Abstract Factory (C)	13. Interpreter (B)*
2. Adapter (S), Adapter (S)*	14. Iterator (B)
3. Bridge (S)	15. Mediator (B)
4. Builder (C)	16. Memento (B)
5. Chains of Responsibility (B)	17. Observer
6. Command (B)	18. Prototype (C)
7. Composite (S)	19. Proxy (S)
8. Decorator (S)	20. Singleton (C)
9. Façade (S)	21. State (B)
10. Factory Method (C)*	22. Strategy (B)
11. Flyweight (S)	23. Template Method (B)*
12. Visitor (B)	

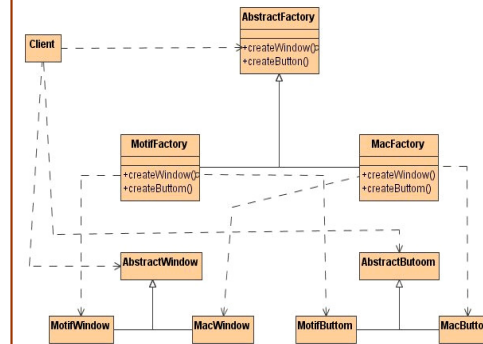
C – creational pattern, S – structural pattern, B – Behavioural pattern, \* - Class Scope

e-Macao-18-1-513

## Applying Abstract Factory

Problem:  
encapsulating  
platforms

Solution:  
**Abstract Factory** -  
shields an  
application from  
the concrete  
classes provided  
by a specific  
platform

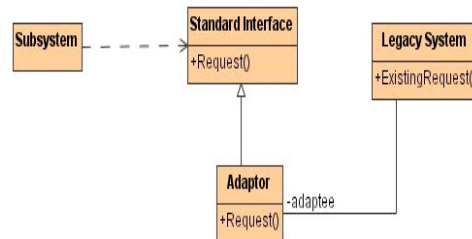


e-Macao-18-1-514

## Applying Adapter

Problem:  
wrapping  
around legacy  
code

Solution:  
**Adapter** -  
encapsulate a  
piece of legacy  
code not  
designed to  
work with the  
system

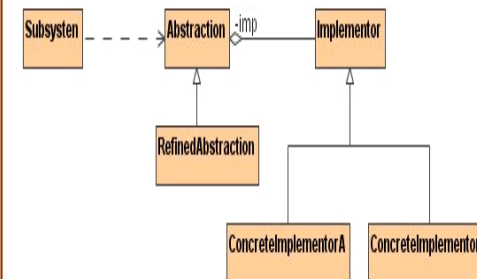


e-Macao-18-1-515

## Applying Bridge

Problem:  
allowing for  
alternate  
implementation

Solution:  
**Bridge** -  
decouples the  
interface from  
the its  
implementation

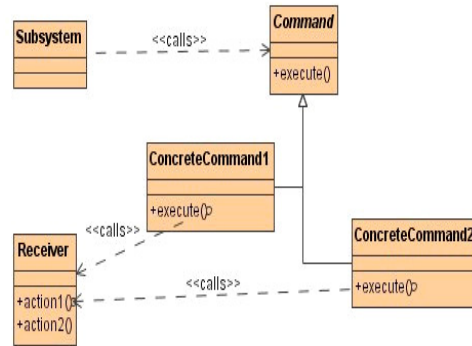


e-Macao-18-1-516

## Applying Command

Problem:  
encapsulating  
control

Solution:  
**Command** -  
encapsulates a  
control such  
that user  
requests can be  
treated  
uniformly

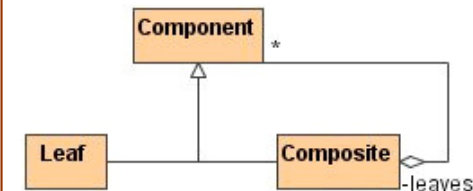


e-Macao-18-1-517

## Applying Composite

Problem:  
representing  
recursive  
hierarchies

Solution:  
**Composite** -  
represent  
recursive  
hierarchies

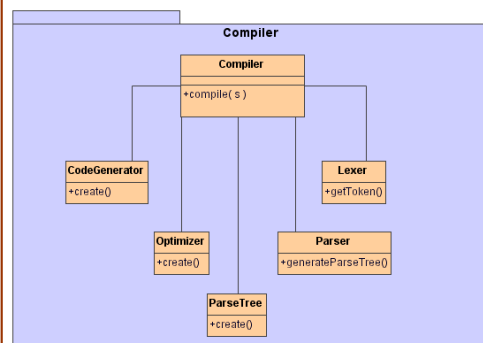


e-Macao-18-1-518

## Applying Façade

Problem:  
encapsulating  
subsystems

Solution:  
**Façade** -  
reduces  
dependencies  
among classes  
by encapsulating  
subsystems from  
with simple  
unified interfaces

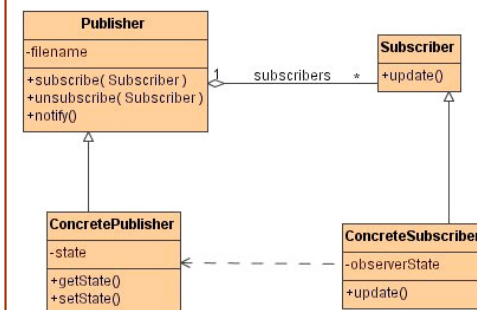


e-Macao-18-1-519

## Applying Observer

Problem:  
decoupling  
entities from  
view

Solution:  
**Observer** -  
allows us to  
maintain  
consistency  
across the state  
of one publisher  
and many  
subscribers



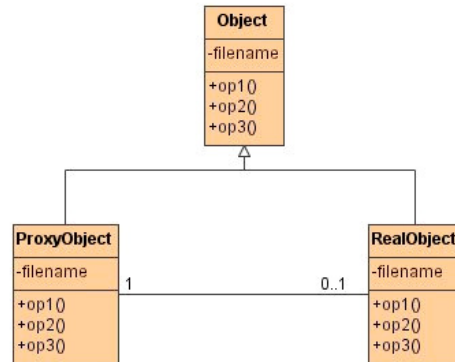


e-Macao-18-1-520

## Applying Proxy

Problem:  
encapsulating  
expensive  
objects

Solution:  
**Proxy** –  
improves the  
performance or  
the security of a  
system by  
delaying  
expensive  
computations, ...  
until when  
needed

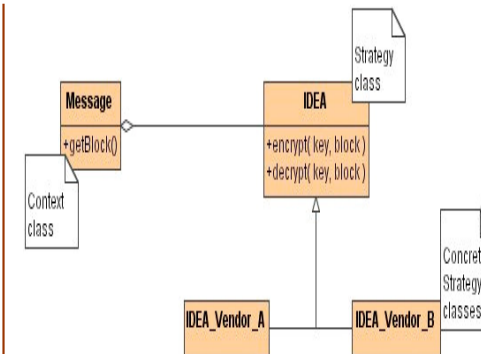


e-Macao-18-1-521

## Applying Strategy

Problem:  
encapsulating  
algorithms

Solution:  
**Strategy** –  
decouples an  
algorithm from  
its  
implementations



e-Macao-18-1-522

## Task 53

For each of the following statements, specify the most convenient pattern to apply to the following scenarios:

- 1) a component on the web tier requires access to business components
- 2) there a need to provide several buttons on a web form which executes different actions
- 3) messages need to be sent to citizens each time a typhoon approaches

e-Macao-18-1-523

## Summary 1

- 1) design involves the transformation of analysis models in the problem space into design models in the solution space
- 2) object design includes:
  - a) service specification for classes
  - b) component selection
  - c) object model restructuring
  - d) object model optimization

e-Macao-18-1-524

## Summary 2

---

- 3) design class diagrams provide specification for software classes and interfaces in an application
- 4) typical information contained in a Design Class Diagram include:
  - a) classes
  - b) associations
  - c) attributes
  - d) interfaces
  - e) methods
  - f) attribute type information
  - g) navigability
  - h) dependencies

e-Macao-18-1-525

## Summary 3

---

- 5) activity diagrams models the dynamic aspect of a system by showing the flow from one activity to another
- 6) activity diagram can be used to describe a workflow or the details of an operation
- 7) an activity diagram may also show the flow of an object as it moves from one state to another at different points in the flow of control

e-Macao-18-1-526

## Summary 4

---

- 9) design sequence diagrams specifies message types, temporal constraints, object creation and destruction
- 10) design statechart diagrams describes detailed the internal behavior of objects
- 11) design patterns are partial solutions to common problems. They name, abstract and identify the key aspects of common design structure that make them useful for creating reusable object oriented design
- 12) there are 23 basic patterns as provided by the GoF

## A.7. Implementation

# Implementation Model

e-Macao-18-1-528

## Course Outline

1) Object Orientation

2) UML Basics

3) UML Modelling:

a) Requirements

b) Architecture

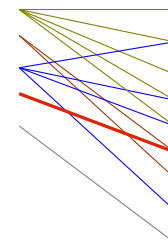
c) Design

d) Implementation

e) Deployment

4) Unified Process

5) UML Tools



UML Diagrams:

1. use case

2. class

3. object

4. sequence

5. state

6. component

7. collaboration

8. activity

9. deployment

e-Macao-18-1-529

## Implementation Phase

At this stage it is necessary to **implement** the design specified in the design models.

It also deals with **non-functional requirements** and the **deployment** of the executable modules onto nodes.

Two models are developed during this phase:

- 1) the **implementation model** describing how the design elements have been implemented in terms of software system
- 2) the **deployment model** describing how the implemented software should be deployed on the physical hardware

e-Macao-18-1-530

## Implementation Model

The implementation model indicates how various aspects of the design map onto the target language.

It describes how components, interfaces, packages and files are related.

The modelling elements are:

- 1) packages
- 2) components
- 3) their relationships

and they are shown in the **implementation component diagram**.

e-Macao-18-1-531

## Modelling Techniques

Component diagrams are used to model the static implementation view of a system.

To model this view, it is possible to use component diagrams in one of four ways:

- 1) to model source code
- 2) to model executable releases
- 3) to model physical databases
- 4) to model adaptable systems

e-Macao-18-1-532

## Modelling Source Code

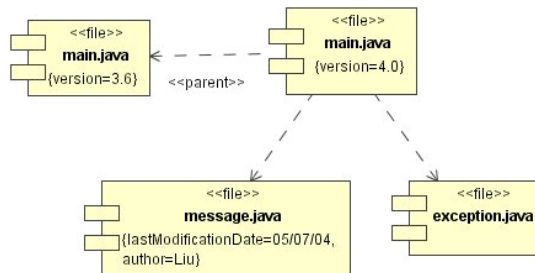
1) Component diagrams are used to model the configuration management of source files, which represent work-product components.

2) Modelling procedure:

- a) identify the set of source code files of interest and model them as components stereotyped as **files**
- b) for larger systems, use packages to show groups of source code files
- c) consider exposing a tagged value to show interest information such as author, version number, etc.
- d) model the compilation dependencies among these files using dependencies

e-Macao-18-1-533

## Source Code Files Example



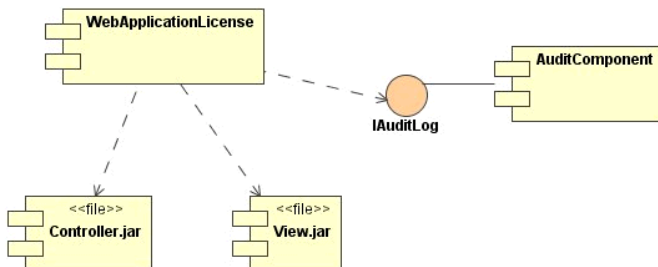
e-Macao-18-1-534

## Modelling Executable Releases

- 1) Component diagrams are used model executable releases, including the deployment components that form each release, and the relationships among those components.
- 2) Each component diagram focuses on one set of components at a time, such as all components that live on one node.
- 3) Modelling procedure:
  - a) identify the set of components to model
  - b) consider the stereotype of each component in the set
  - c) for each component, consider its relationship to its neighbors.

e-Macao-18-1-535

## Executables Example



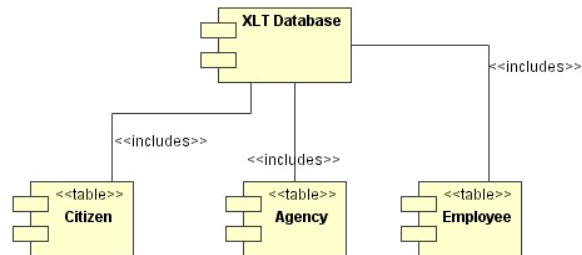
e-Macao-18-1-536

## Modelling Physical Databases

- 1) Component diagrams model the mapping of classes into tables of a database.
- 2) Modelling procedure:
  - a) identify the classes in your model that represent your logical database schema
  - b) select a strategy for mapping these classes to tables, possibly considering the physical distribution
  - c) create a component diagram containing components stereotyped as **tables** to model the mapping

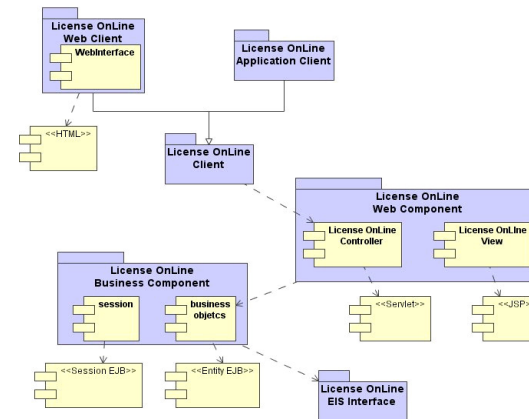
e-Macao-18-1-537

## Physical Databases Example



e-Macao-18-1-538

## Example



e-Macao-18-1-539

## Summary

- 1) The implementation component diagram describes how components, interfaces, packages and files are related.
- 2) Implementation component diagrams may be used to model source code, executable releases, physical databases and adaptable systems.

e-Macao-18-1-540

## Task 54

Provide an possible implementation model for the restaurant license application based on any implementation framework of your choice (.NET, J2EE, etc.). Your model should show the major components of the system.

**A.8. Deployment**

e-Macao-18-1-541

## Deployment Model

e-Macao-18-1-542

### Course Outline

1) Object Orientation

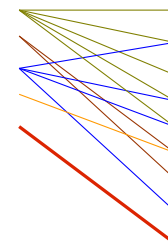
2) UML Basics

3) UML Modelling:

- a) Requirements
- b) Architecture
- c) Design
- d) Implementation
- e) Deployment

4) Unified Process

5) UML Tools



UML Diagrams:

- 1. use case
- 2. class
- 3. object
- 4. sequence
- 5. state
- 6. component
- 7. collaboration
- 8. activity
- 9. deployment

e-Macao-18-1-543

## Deployment Diagrams 1

- 1) show the configuration of run-time processing **nodes** and the **components** that live on them.
- 2) model the distribution, delivery, and installation of the parts that make up the physical system
- 3) involve modelling the topology of the hardware on which the system executes

e-Macao-18-1-544

## Deployment Diagrams 2

- 4) essentially focus on a system's nodes, and include:
  - a) nodes
  - b) dependencies and associations relationships
  - c) components
  - d) packages

e-Macao-18-1-545

## Nodes

- 1) model the topology of the hardware on which the system executes
- 2) represent the hardware on which components are deployed and executed
- 3) may be stereotyped to allow for specific kinds of processors and devices.

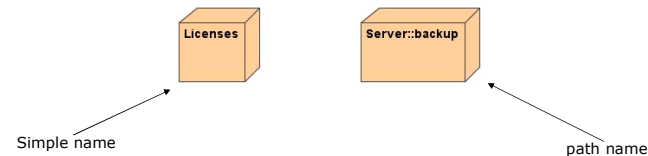


e-Macao-18-1-546

## Nodes Names

Every node must have a name that distinguishes it from other nodes.

A name is a textual string which may be written as a simple name or as a path name.





e-Macao-18-1-547

## Nodes and Components

### Components

- 1) participate in the execution of a system.
- 2) represent the physical packaging of otherwise logical elements

### Nodes

- 1) execute components
- 2) represent the physical deployment of components

The relationship **deploys** between a node and a component can be shown using a **dependency relationship**.

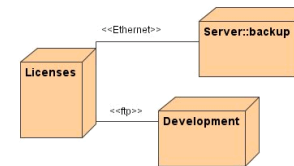
e-Macao-18-1-548

## Organizing Nodes

Nodes can be organized:

- 1) in the same manner as classes and components
- 2) by specifying dependency, generalization, association, aggregation, and realization **relationships** among them.

The most common kind of relationship used among nodes is an **association** representing a physical connection among them.

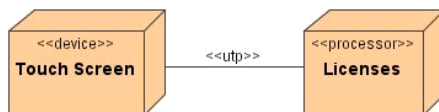


e-Macao-18-1-549

## Processors and Devices

A **processor** is a node that has processing capability. It can execute a component.

A **device** is a node that has no processing capability (at least at the level of abstraction showed).



e-Macao-18-1-550

## Modelling Nodes

Procedure:

- 1) identify the computational elements of the system's deployment view and model each as a node
- 2) add the corresponding stereotype to the nodes
- 3) consider attributes and operations that might apply to each node.

e-Macao-18-1-551

## Distribution of Components

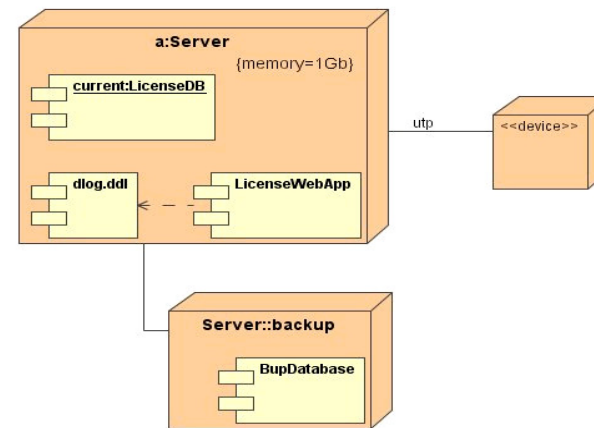
To model the topology of a system it is necessary to specify the physical distribution of its components across the processors and devices of the system.

Procedure:

- 1) allocate each component in a given node
- 2) consider duplicate locations for components, if it is necessary
- 3) render the allocation in one of these ways:
  - a) don't make visible the allocation
  - b) use dependency relationship between the node and the component it's deploy
  - c) list the components deployed on a node in an additional compartment

e-Macao-18-1-552

## Example



e-Macao-18-1-553

## Summary

Deployment diagrams model the topology of the hardware on which the system executes and the software installed on each of the hardware components.

Deployment diagrams include nodes, packages, components and their relationships.

Deployment diagrams may be used to model different kind of systems such as embedded, client-server or distributed systems.

e-Macao-18-1-554

## Task 55:

Consider task 54, show how these components will be deployed at the agency. In your model, annotate the nodes according to described them.

**A.9. Unified Process**

e-Macao-18-1-555

# Unified Process

e-Macao-18-1-556

## Course Outline

1) Object Orientation

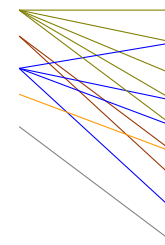
2) UML Basics

3) UML Modelling:

- a) Requirements
- b) Architecture
- c) Design
- d) Implementation
- e) Deployment

4) Unified Process

5) UML Tools



UML Diagrams:

- 1. use case
- 2. class
- 3. object
- 4. sequence
- 5. state
- 6. component
- 7. collaboration
- 8. activity
- 9. deployment

e-Macao-18-1-557

## Overview

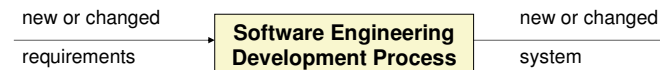
- 1) Software Development Process
- 2) Unified Process Overview
- 3) Unified Process Structure
  - 1) Building Blocks
  - 2) Phases
  - 3) Workflows

e-Macao-18-1-558

## Why a process ?

A process defines:

- 1) **who** is doing **what**
- 2) **when** to do it
- 3) **how** to reach a certain goal
- 4) the inputs and outputs for each activity



e-Macao-18-1-559

## Development Process

Is a **framework** which guides the tasks, people and define output products of the development process.

It is a framework because:

- 1) provides the inputs and outputs of each activity
- 2) does not restrict how each activity must be performed
- 3) must be tailored for every project

There is **no universal process**.

e-Macao-18-1-560

## Unified Process - Overview

Key elements:

- 1) iterative and incremental
- 2) use case-driven
- 3) architecture-centric

e-Macao-18-1-561

## Iterative and Incremental 1

The design process is based on iterations that address different aspects of the design process.

The iterations evolve into the final system (incremental aspect).

The process does not try to complete the whole design task in one go.

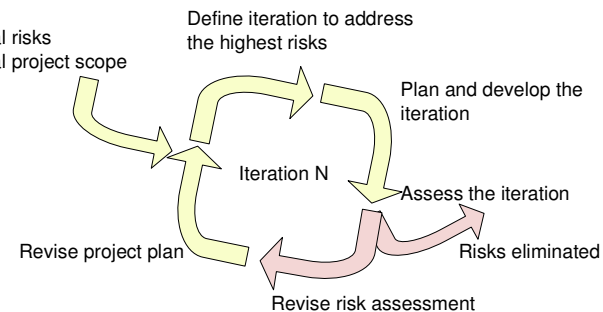
How to do it?

- 1) plan a little
- 2) specify, design and implement a little
- 3) integrate, test and run
- 4) obtain feedback before next iteration

e-Macao-18-1-562

## Iterative and Incremental 2

Technical risks are assessed and prioritized early and are revised during each iteration.



e-Macao-18-1-563

## Use Case-Driven

Use cases are used for:

- 1) identify users and their requirements
- 2) aid in the creation and validation of the architecture
- 3) help produce definitions of test cases and procedures
- 4) direct the planning of iterations
- 5) drive the creation of user documentation
- 6) direct the deployment of the system
- 7) synchronize the content of different models
- 8) drive traceability throughout models

e-Macao-18-1-564

## Architecture-Centric

**Problem:**

- with the iterative and incremental approach different development activities are done concurrently

**Solution:**

- the system's architecture ensures that all parts fit together

"An architecture is the skeleton on which the muscles (functionality) and skin (user-interface) of the system will be hung".

e-Macao-18-1-565

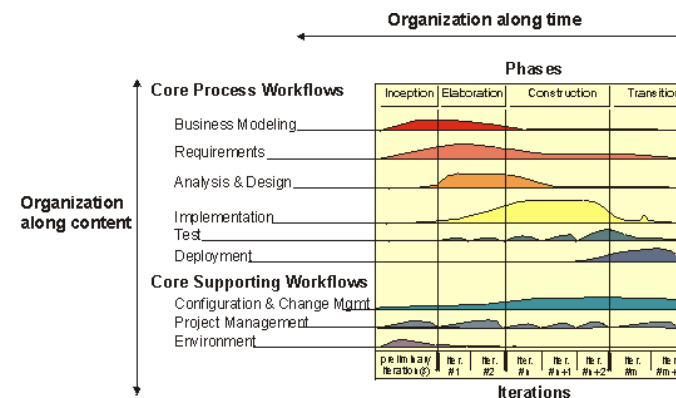
## Unified Process - Structure

Two dimensions:

- 1) **time** → division on the life cycle into **phases** and iterations
- 2) **process components** → production of a specific set of artifacts with well-defined activities called **workflows**

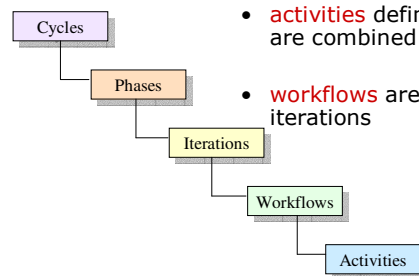
e-Macao-18-1-566

## The Development Process



e-Macao-18-1-567

## Building Blocks

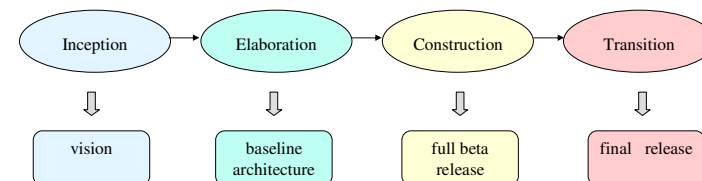


- **activities** define detailed work and are combined in workflows
- **workflows** are organized into iterations
- each **iteration** identifies some aspect of the system and are organized into phases
- **phases** can be grouped into cycles
- **cycles** focus on the generation of successive releases

e-Macao-18-1-568

## Life Cycle Phases

**Phases** and **major deliverables** of the Unified Process



e-Macao-18-1-569

## Inception

Focuses on the generation of the business case that involves:

- 1) identification of core uses cases
- 2) definition of the actual scope
- 3) identification of risky difficult parts of the system

The main objectives are:

- 1) to prove the feasibility of the system to be built
- 2) to determine the complexity involved in order to provide reasonable estimates

Outputs:

- 1) **the vision of the system**
- 2) very simplified use case model
- 3) tentative architecture
- 4) risks identified
- 5) plan for the elaboration phase

e-Macao-18-1-570

## Elaboration

Involves:

- 1) understanding how requirements are translated into the internals of the system
- 2) producing the baseline architecture
- 3) capturing the majority of the use cases
- 4) exploring further the risks identified earlier and identifying the most significant
- 5) specifying any non-functional requirements specially those related to reliability and performance

Outputs:

- 1) **the system's architecture**
- 2) detailed use case model
- 3) set of plans for the construction phase

e-Macao-18-1-571

## Construction

Involves:

- 1) completing the analysis of the system
- 2) performing the majority of the design and the implementation

Outputs:

- 1) **implemented software product as a full beta release.**  
It may contain some defects
- 2) associated models

An important aspect for the success of this phase is to monitor the critical aspects of the projects, specially significant risks.

e-Macao-18-1-572

## Transition

Involves:

- 1) deployment of the beta system
- 2) monitoring user feedback and handling any modifications or updates required

Output:

- 1) **the formal release of the software**

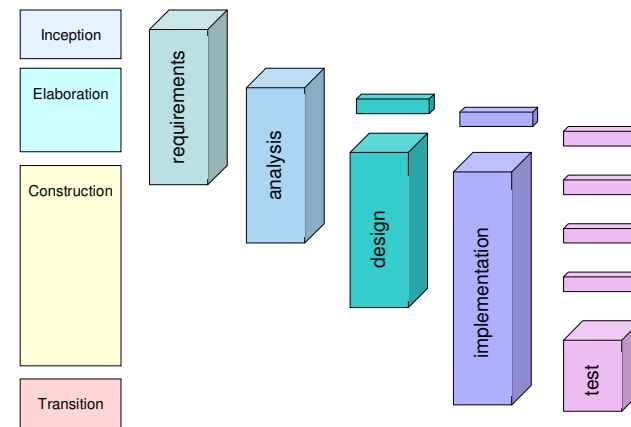
e-Macao-18-1-573

## UP - Workflows 1

- 1) **requirements**: focuses on the activities which allow to identify functional and non-functional requirements
- 2) **analysis**: restructures the requirements identified in terms of the software to be built
- 3) **design**: produces a detailed design
- 4) **implementation**: represents the coding of the design in a programming language, and the compilation, packaging, deployment and documentation of the software
- 5) **test**: describes the activities to be carried out for testing

e-Macao-18-1-574

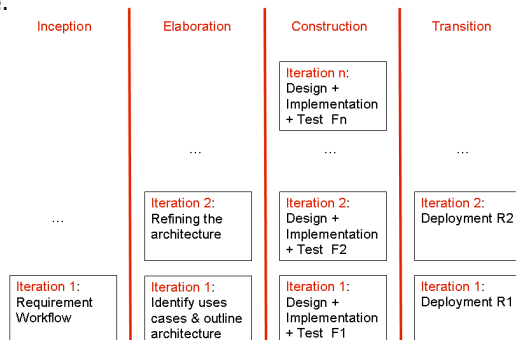
## Workflows and Phases



e-Macao-18-1-575

## Phases and Iterations

The iterations of workflows occur one or more times during a phase.



e-Macao-18-1-576

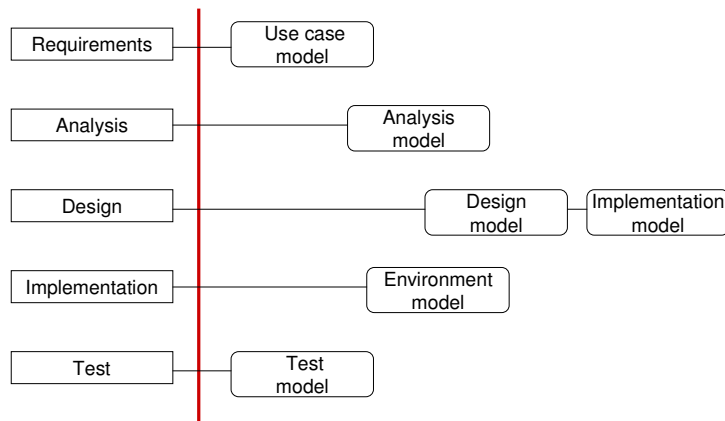
## Workflows and Activities

Workflows	Activities
Requirements	Find actors and use cases, prioritize use cases, detail use cases, prototype user interface, structure the use case model
Analysis	Architectural analysis, analyze use cases, explore classes, find packages
Design	Architectural design, trace use cases, refine and design classes, design packages
Implementation	Architectural implementation, implement classes and interfaces, implement subsystems, perform unit testing, integrate systems
Test	Plan and design tests, implement tests, perform integration and system tests, evaluate tests



e-Macao-18-1-577

## Workflows and Models



e-Macao-18-1-578

## Summary 1

A software development process is a framework that provides guidance to carry out the different activities needed to produce a software product.

Every software development process must be parameterized for each individual project.

There is no universal process.

e-Macao-18-1-579

## Summary 2

The main features of the Unified Process are:

- 1) use case-driven
- 2) architecture-centric
- 3) iterative and incremental

With respect to time dimension, the lifecycle is divided into phases and iteration.

There are four main phases: inception, elaboration, construction and transition.

A specific set of artifacts are produced with well-defined activities called workflows.

There are five main workflows: requirements, analysis, design, implementation and test.

**A.10. UML Tools**

# UML Tools

e-Macao-18-1-581

## Course Outline

1) Object Orientation

2) UML Basics

3) UML Modelling:

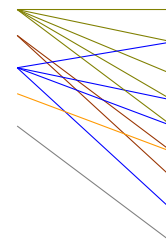
- a) Requirements
- b) Architecture
- c) Design
- d) Implementation
- e) Deployment

4) Unified Process

5) UML Tools

UML Diagrams:

- 1. use case
- 2. class
- 3. object
- 4. sequence
- 5. state
- 6. component
- 7. collaboration
- 8. activity
- 9. deployment



e-Macao-18-1-582

## Overview

- 1) Why UML CASE tools?
- 2) Benefits
- 3) Different tools
- 4) Evaluating UML CASE tools
- 5) Main Features of:
  - a) Enterprise Architect
  - b) MagicDraw
  - c) Poseidon
  - d) Rational Rose

e-Macao-18-1-583

## Why UML CASE Tools

- 1) tools offer benefits to everyone involved in a project:
  - a) **analysts** can capture requirements with use case model
  - b) **designers** can produce models that capture interactions between objects
  - c) **developers** can quickly turn the model into a working application
- 2) UML case tool, plus a methodology, plus empowered resources enable the development of the right software solution, faster and cheaper

e-Macao-18-1-584

## Different UML CASE Tools

Tools vary with respect to:

- 1) UML modelling capabilities
- 2) project life-cycle support
- 3) forward and reverse engineering
- 4) data modelling
- 5) performance
- 6) price
- 7) supportability
- 8) easy of use
- 9) ...

e-Macao-18-1-585

## Evaluating CASE Tools

Different Criteria for evaluating CASE tools:

- 1) repository support
- 2) round-trip engineering
- 3) HTML documentation
- 4) UML support
- 5) data modelling integration
- 6) versioning
- 7) model navigation
- 8) printing support
- 9) diagrams views
- 10) exporting diagrams
- 11) platform

e-Macao-18-1-586

## Different UML Tools

- **Enterprise Architect**
  - organization: Sparx Systems
  - web-site: <http://www.sparxsystems.com.au/>
- **MagicDraw**
  - organization: No Magic Inc.
  - web-site: <http://www.nomagic.com/>
- **Poseidon**
  - organization: Gentleware
  - web-site: <http://www.gentleware.com/>
- **Rational Rose**
  - organization: IBM
  - web-site: <http://www-306.ibm.com/software/rational/>

e-Macao-18-1-587

## Enterprise Architect

Criteria	Features
Platform	Windows
UML Compliance	Support for all 13 UML 2.0 diagrams
Usability	Replication capable – Comprehensive and flexible documentation
Development Environment	Multi-user enabled – Allows to replicate and share projects
Outputs & Code Generation	C++, Java, C#, VB, VB.Net, Delphi, PHP – HTML and RTF document generation - Forward and reverse database engineering
Data Repository	Models are stored in a data repository - Checks data integrity in the data repository – Provides a project browser
Other features	Allows scripting to extend functionality – Project estimation tools – User definable patterns

e-Macao-18-1-588

## Magic Draw

Criteria	Features
Platform	Any where Java 1.4 is supported
UML Compliance	Support for UML 1.4 notation and semantics
Usability	Replication capable – Customizable views of UML elements – Customizable elements properties
Development Environment	Multi-user enabled – Lock parts of the model to edit – Commit changes – Model versioning and rollback
Outputs & Code Generation	Code generation and reverse engineering to C++, Java, C# - RTF and PDF document generation
Data Repository	Provides a project browser
Other features	Friendly and customizable GUI – Hyperlinks can be added to any model element

e-Macao-18-1-589

## Poseidon

Criteria	Features
Platform	Platform independent – Implemented in Java
UML Compliance	Supports all 9 diagrams of UML 1.4
Usability	Replication capable – Internationalization and localization for several languages
Development Environment	Collaborative environment based on client-server architecture – Locking of model parts – Secure transmission of files
Outputs & Code Generation	VB.Net, C#, C++, CORBA IDL, Delphi, Perl, PHP, SQL DDL – Round trip engineering for Java – Diagram export as gif, ps, eps and svg.
Data Repository	Uses MDR (Meta Data Repository) developed by Sun and based on the JMI (Java Metadata Interface) standard
Other features	Allows to import Rational Rose files

e-Macao-18-1-590

## Rational Rose

Criteria	Features
Platform	Windows
UML Compliance	Not fully supported UML 1.4
Usability	The add-in feature allows to customize the environment – User configurable support for UML, OMT and Booch 93.
Development Environment	Parallel multi-user development through repository and private support
Outputs & Code Generation	C++, Visual C++, VB6, Java – Documentation generation – Round trip engineering
Data Repository	Maintains consistency between the diagram and the specification, you may change any of them and automatically updates the information.
Other features	Can be integrated with other Rational products such as RequisitePro, Test Manager

e-Macao-18-1-591

## Summary

There exist several CASE Tools supporting Object Oriented modelling with UML.

Different criteria should be considered when evaluating a software tool.

Four tools were presented: Enterprise Architect, Magic Draw, Poseidon and Rational Rose.

**A.11. Summary**

## Course Summary

e-Macao-18-1-593

## Object Orientation

**Object Orientation** is about viewing and modelling the world/system as a set of interacting and interrelated *objects*.

Four principles:

- 1) abstraction
- 2) encapsulation
- 3) modularity
- 4) hierarchy

e-Macao-18-1-594

## Object Oriented Concepts

- Object
- Class
- Attribute
- Operation
- Interface
- Implementation
- Association
- Aggregation
- Composition
- Generalization
- Super-class / Sub-class
- Abstract class / Concrete-class
- Discriminator
- Polymorphism
- Realization

e-Macao-18-1-595

## UML Basics

Unified Modeling Language – standard published by OMG

Building Blocks:

- 1) elements
- 2) relationships
- 3) diagrams

e-Macao-18-1-596

## UML Elements 1

Structural:

- 1) class
- 2) interface
- 3) collaboration
- 4) use case
- 5) active class
- 6) component
- 7) node

e-Macao-18-1-597

## UML Elements 2

Behavioural:

- 1) interactions
- 2) state-machines

Grouping:

- 1) package

Annotation:

- 1) notes

e-Macao-18-1-598

## UML - Diagrams

### Static

- 1) Class
- 2) Object
- 3) Component
- 4) Deployment

### Dynamic

- 1) Use Case
- 2) Sequence
- 3) Collaboration
- 4) Statechart
- 5) Activity

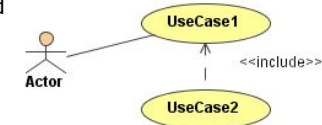
e-Macao-18-1-599

## Use Case Diagram

Shows the functions of the system (external view).

Components:

- 1) use cases
- 2) actors
- 3) relationships:
  - a) use cases:
    - generalization
    - dependency: include
  - b) actors:
    - generalization
  - c) use cases – actors:
    - association



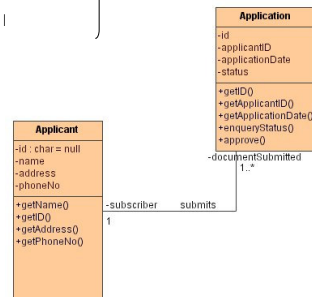
e-Macao-18-1-600

## Class Diagram

Shows the classes and their relationships.

Main components:

- 1) classes
    - attributes – types, default value
    - operations – arguments, return type
  - 2) relationships
    - association
    - aggregation
    - composition
    - generalization
    - dependency
- visibility
- multiplicity
- roles



e-Macao-18-1-601

## Object Diagram

Shows the instances for a given class diagram in a point of time.

Components:

- 1) objects
- 2) links





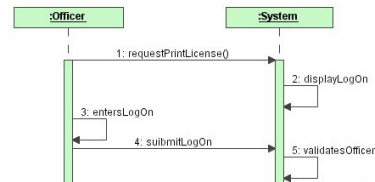
e-Macao-18-1-602

## Sequence Diagram

Shows interactions between objects – (ordered in time).

Main components:

- 1) object lifelines:
  - object
  - lifeline
- 2) interactions - messages



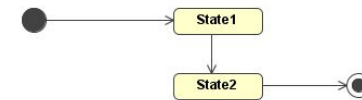
e-Macao-18-1-603

## Statechart Diagram

Shows the state of an object(s) in response to events.

Main components:

- 1) states:
  - pseudo-states:
    - initial – final
    - static/dynamic branch points
  - simple states
  - composite states
    - mutually exclusive sub-states
    - concurrent sub-states
- 2) events



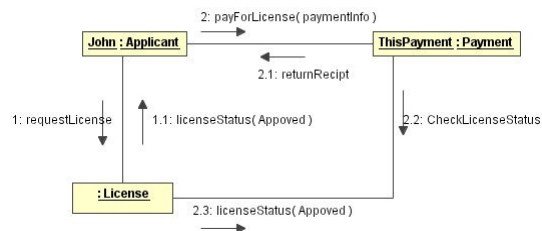
e-Macao-18-1-604

## Collaboration Diagrams

Shows interaction between objects – (ordered in time).

Main components:

- a) classes – associations
  - objects – links
- b) messages



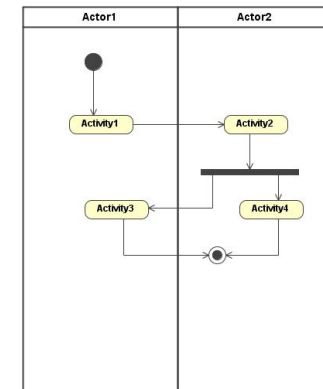
e-Macao-18-1-605

## Activity Diagram

Focuses on the internal execution of activities.

Main components:

- 1) states
- 2) transitions
- 3) decisions
- 4) forks – joins
- 5) swimlanes



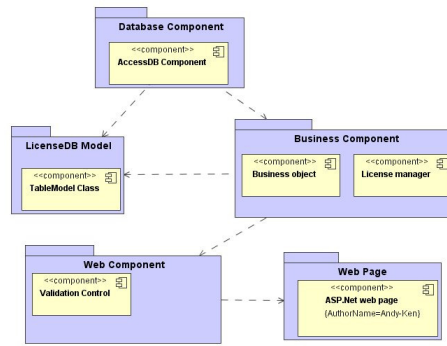
e-Macao-18-1-606

## Component Diagram

Shows structural replaceable parts of the system.

Main components:

- 1) components
- 2) interfaces
- 3) packages



e-Macao-18-1-608

## Development Stages

- 1) Requirements
  - 2) Analysis
  - 3) Design → architecture and detailed design
  - 4) Implementation
  - 5) Deployment
- } requirements modelling

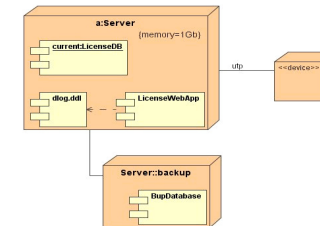
e-Macao-18-1-607

## Deployment Diagram

Shows the nodes and components of the system.

Main components:

- 1) nodes:
  - processor
  - devices
- 2) components
- 3) packages



e-Macao-18-1-609

## Development 1

- Requirements Modelling:
  - functional and non-functional requirements
  - template and glossary
  - diagrams:
    - 1) use case diagram
    - 2) class diagram
    - 3) sequence diagram
    - 4) statechart diagram

e-Macao-18-1-610

## Development 2

- **Architecture:**
  - concepts:
    - subsystem
    - cohesion – coupling
    - layering
  - diagrams:
    - collaboration diagram
    - component diagram
  - architecture styles:
    - repository
    - MVC
    - client-server

e-Macao-18-1-611

## Development 3

- **Design:**
  - activities
  - diagrams:
    - class diagrams
    - activity diagrams
    - sequence diagrams
    - statecharts diagrams
  - patterns

e-Macao-18-1-612

## Development 4

- **Implementation:**
  - diagram:
    - component diagram

e-Macao-18-1-613

## Development 5

- **Deployment:**
  - diagram:
    - deployment diagram

e-Macao-18-1-614

## Unified Process

- 1) iterative and incremental
- 2) use case-driven
- 3) architecture-centric

e-Macao-18-1-615

## UP Workflows

- 1) Requirements
- 2) Analysis
- 3) Design
- 4) Implementation
- 5) Test

e-Macao-18-1-616

## UML Tools

- Magic Draw
- Rational Rose
- Poseidon
- Enterprise Architect

The End

e-Macao-18-1-618

## Acknowledgements

We would like to thank Dr. Tomasz Janowski and all the members of the eMacao team for their valuable comments and help in preparing this material.

## B. Assessment

### B.1. Set 1

1	Divide a complex system into small, self-contained pieces that can be managed independently. How is it called?	
	a	Abstraction
	b	Modularity
	c	Encapsulation
	d	Hierarchy
	Answer: B	

2	In order to model the relationship "a course is composed of 5 to 20 students and one or more instructors", you could use:	
	a	Aggregation
	b	Association
	c	Composition
	d	Realization
	Answer: A	

3	Which of the following statements are true?	
	a	All operations defined in a sub-class are inherited by the super-class
	b	Generalization allows abstracting common features and defining them in a super-class
	c	A super-class is a class that must not have associations
	d	Association is a "part-of" relationship
	Answer: B	

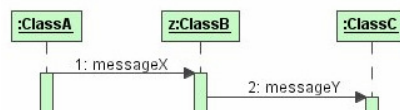
4	What is the relationship between these two use cases?	
	a	Generalization
	b	Extend
	c	Include
	d	Association
	Answer: C	

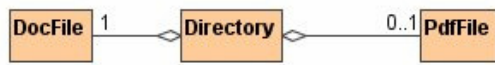
5	The following diagram shows that there is an interaction between:	
	a	An object of ClassA and an object of ClassB
	b	An object of ClassA and object z of ClassB
	c	Object z of ClassB and an object of ClassB
	d	An object of ClassC and an object of ClassA
	Answer: B	

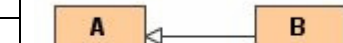
SubmitLicense  
Application

SubmitDriving  
LicenseApplication



6	Which of the following statements are true for the following diagram?			
	a	State1 is always the first state of the object after the initial state	c	State3 is always the last state of the object before the final state
	b	State2 is always the last state of the object before the final state	d	During its life, the object is in at least five states
	Answer: C			

7	How many <i>Files</i> objects for each <i>Directory</i> object?		
	a	0 or 1	
	b	2	
	c	1 or 2	
	d	Many	
	Answer: C		

8	Which of the following statement is true for the following diagram		
	a	A is a kind of B	
	b	B is a kind of B	
	c	A is part of B	
	d	B depends on A	
	Answer: None		

9	Which of these diagrams shows interactions between objects?		
	a	Activity diagram	
	b	Class diagram	
	c	Sequence diagram	
	d	Component diagram	
	Answer: C		

10	A statechart diagram describes:		
	a	Attributes of objects	
	b	Nodes of the system	
	c	Operations executed on a thread	
	d	Events triggered by an object	
	Answer: D		

11	An interface is:		
	a	A set of objects used to provide a specific behaviour	
	b	A set of classes used on a collaboration	
	c	A set of attributes used on an operation	
	d	A set of operations used to specify a service of a class or component	
	Answer: D		

12	The sequence diagram models:
a	The order in which the class diagram is constructed
b	The way in which objects communicate
c	The relationship between states
d	The components of the system
	Answer: B
13	The activity diagram:
a	Focuses on flows driven by internal processing
b	Models the external events stimulating one object
c	Focuses on the transitions between states of a particular object
d	Models the interaction between objects
	Answer: A
14	The deployment diagram shows:
a	Objects of a system
b	Distribution of components on the nodes in a system
c	Functions of a system
d	Distribution of nodes
	Answer: B
15	Unified Process is a software development methodology which is:
a	Use-case driven
b	Component-driven
c	Related to Extreme Programming
d	None in only one iteration
	Answer: A



**B.2. Set 2**

1	Ordering abstractions into a tree-like structure. How is it called?
a	Abstraction
b	Modularity
c	Encapsulation
d	Hierarchy
	Answer: D

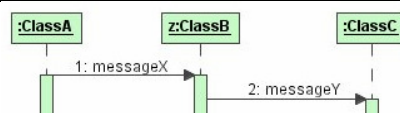
2	In order to model the relationship "a hotel has rooms", between Hotel and Room, you could use:
a	Aggregation
b	Association
c	Composition
d	Realization
	Answer: C

3	Which of the following statements are true?
a	All operations defined in a super-class are inherited by the sub-class
b	Generalization allows abstracting common features and defining them in a sub-class
c	A super-class is a class that must not have associations
d	Association is a "kind-of" relationship
	Answer: A

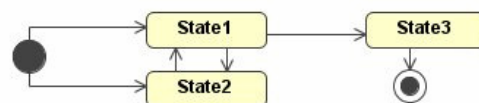
4	What is the relationship between these two use cases?
a	Include
b	Extension
c	Generalization
d	Association
	Answer: C




5	The following diagram shows that there is an interaction between:
a	An object of ClassC and an object of ClassA
b	Object z of ClassB and an object of ClassB
c	An object of ClassA and object z of ClassB
d	An object of ClassA and an object of ClassC
	Answer: C



6	Which of the following statements are true for the following diagram?
a	State2 is always the first state of the object after the initial state
b	State1 is always the last state of the object before the final state
c	State3 is always the last state of the object before the final state
d	During its life, the object is in at least five states
	Answer: C



7	How many <i>Files</i> objects for each <i>Directory</i> object?	
	a	0 or 2
	b	0 or 1
	c	1 or 2
	d	Many
	Answer: C	




```

classDiagram
    DocFile "1" --> "1" Directory
    Directory "1" --> "0..1" PdfFile
  
```

8	Which of the following statement is true for the following diagram	
	a	B is a kind of A
	b	A is part of B
	c	A is a kind of B
	d	B depends on A
	Answer: A	



```

classDiagram
    B --|> A
  
```

9	Which of these diagrams shows interactions between objects?	
	a	Sequence diagram
	b	Class diagram
	c	Activity diagram
	d	Component diagram
	Answer: A	

10	A statechart diagram describes:	
	a	Operations executed on a thread
	b	Nodes of the system
	c	Attributes and operations of an object
	d	Events triggered by an object
	Answer: D	

11	An interface is:	
	a	A set of classes used on a collaboration
	b	A set of operations used to specify a service of a class or component
	c	A set of attributes used on an operation
	d	A set of objects used to provide a specific behaviour
	Answer: B	

12	The sequence diagram models:	
	a	The order in which the class diagram is constructed
	b	The relationship between objects
	c	The way in which objects communicate
	d	The components of the system
	Answer: C	

13	The activity diagram:	
	a	Models the interaction between objects
	b	Models the external events stimulating one object
	c	Focuses on the transitions between states of a particular object
	d	Focuses on flows driven by internal processing
	Answer: D	

14	The deployment diagram shows:
a	Objects of a system
b	Functions of a system
c	Distribution of components on the nodes in a system
d	Distribution of nodes
	Answer: C

15	Unified Process is a software development methodology which is:
a	Component-driven
b	Iterative and incremental
c	Related to Extreme Programming
d	Done in only one iteration
	Answer: B