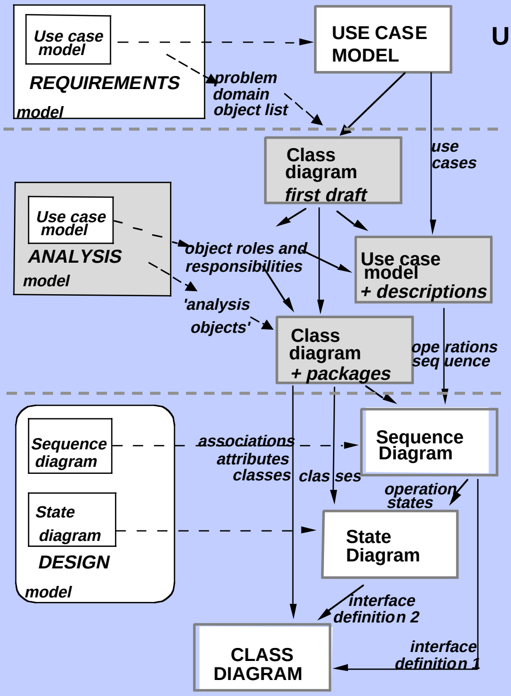
Unit 6: Object-Oriented Software Engineering: Analysis Model

Objectives

This Unit will outline the construction of the *Analysis Model* building on outputs of *Requirements Model.* It will describe the basic *UML notations* associated with analysis and introduce new types of analysis objects . The *use cases* will be used and refined and the inputs for *Design Model* defined.

OOSE Models

UML Representations



Use case model

REQUIREMENTS

model

USE CASE MODEL

problem domain object list

Class diagram *first draft*

use cases

Use case model

ANALYSIS

model

object roles and responsibilities

Use case model *+ descriptions*

'analysis objects'

Class diagram *+ packages*

ope rations seq uence

Sequence diagram

State diagram

DESIGN

model

associations attributes classes

clas ses

Sequence Diagram

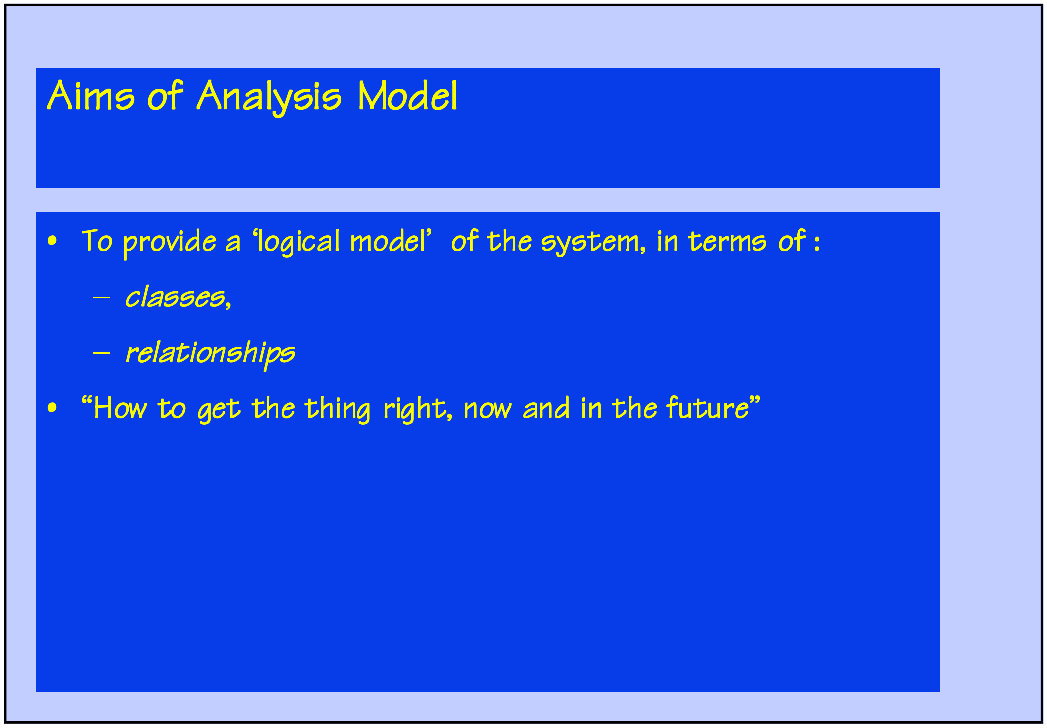
operation states

State Diagram

interface definition 2

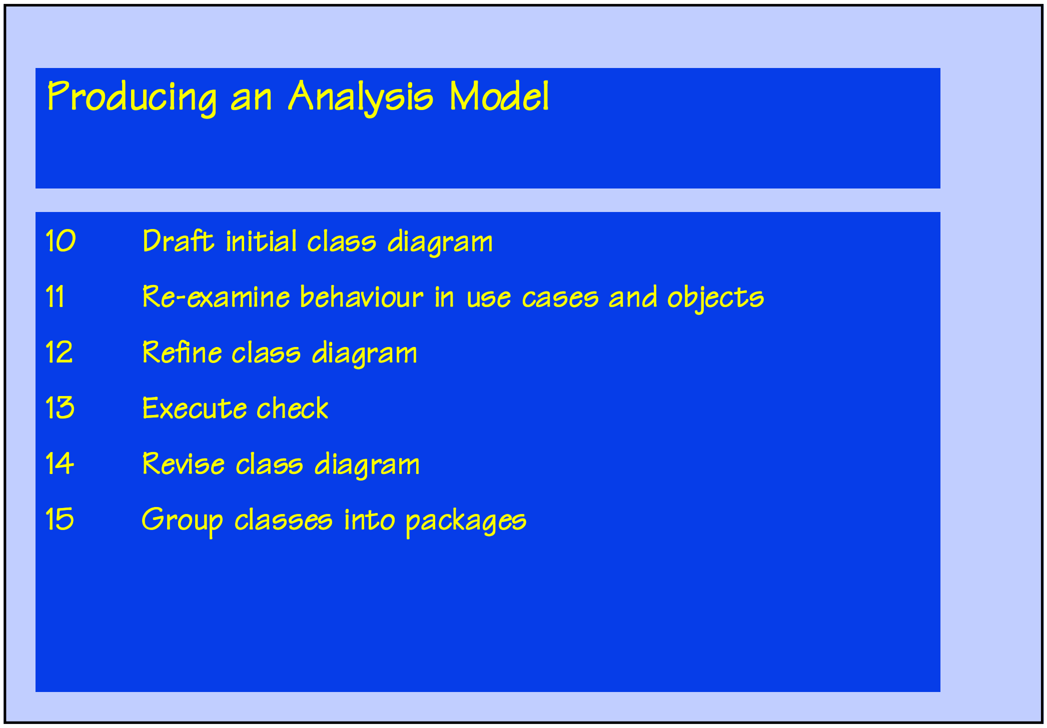
CLASS DIAGRAM

interface definition 1



Aims of Analysis Model

- To provide a 'logical model' of the system, in terms of :
  - *classes,*
  - *relationships*
- "How to get the thing right, now and in the future"



Producing an Analysis Model

10 Draft initial class diagram

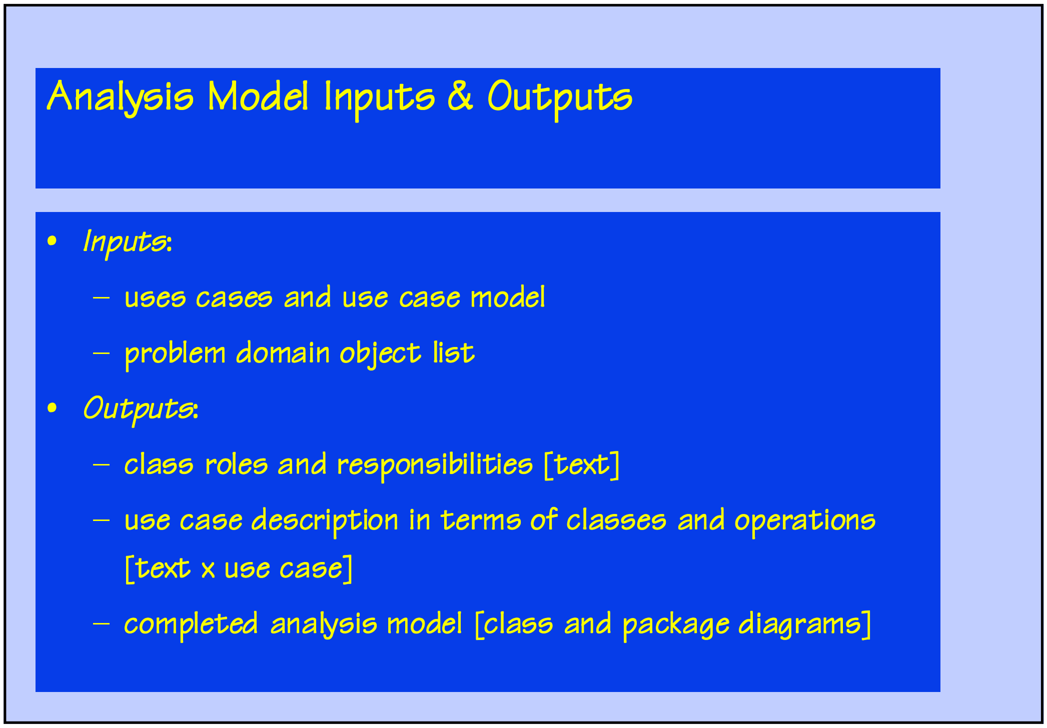
11 Re-examine behaviour in use cases and objects

12 Refine class diagram

13 Execute check

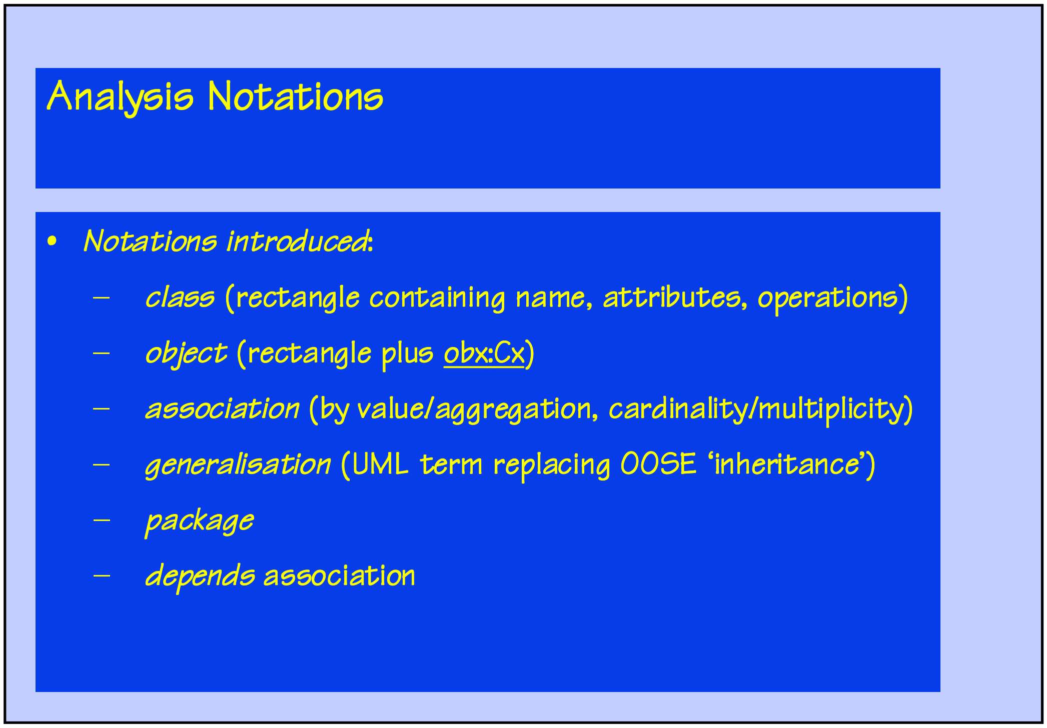
14 Revise class diagram

15 Group classes into packages



Analysis Model Inputs & Outputs

- *Inputs*:
  - uses cases and use case model
  - problem domain object list
- *Outputs*:
  - class roles and responsibilities [text]
  - use case description in terms of classes and operations [text x use case]
  - completed analysis model [class and package diagrams]



Analysis Notations

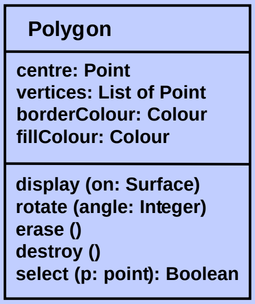
- *Notations introduced*:
  - *class* (rectangle containing name, attributes, operations)
  - *object* (rectangle plus obx:Cx)
  - *association* (by value/aggregation, cardinality/multiplicity)
  - *generalisation* (UML term replacing OOSE 'inheritance')
  - *package*
  - *depends* association

Classes in UML

className

attribute name: type

operation name (parameter: type): result type

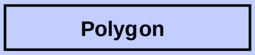


Polygon

centre: Point
vertices: List of Point
borderColour: Colour
fillColour: Colour

display (on: Surface)
rotate (angle: Integer)
erase ()
destroy ()
select (p: point): Boolean

className

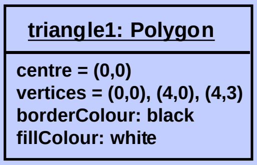


Polygon

Objects in UML

objectName: className

attribute name: type = value



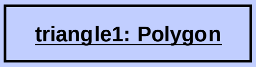
triangle1: Polygon

centre = (0,0)
vertices = (0,0), (4,0), (4,3)
borderColour: black
fillColour: white

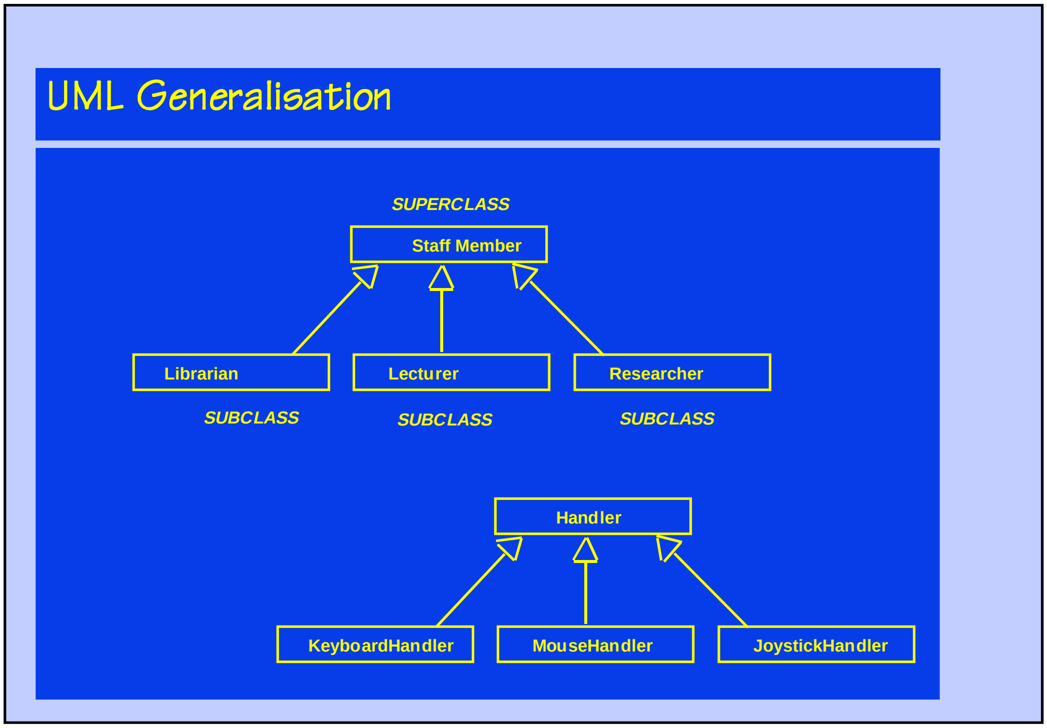
(same operations
for all instances
of a class)

display (on: Surface)
rotate (angle: Integer)
erase ()
destroy ()
select (p: point): Boolean

objectName: className



triangle1: Polygon



UML Generalisation

SUPERCLASS

Staff Member

Librarian

Lecturer

Researcher

SUBCLASS

SUBCLASS

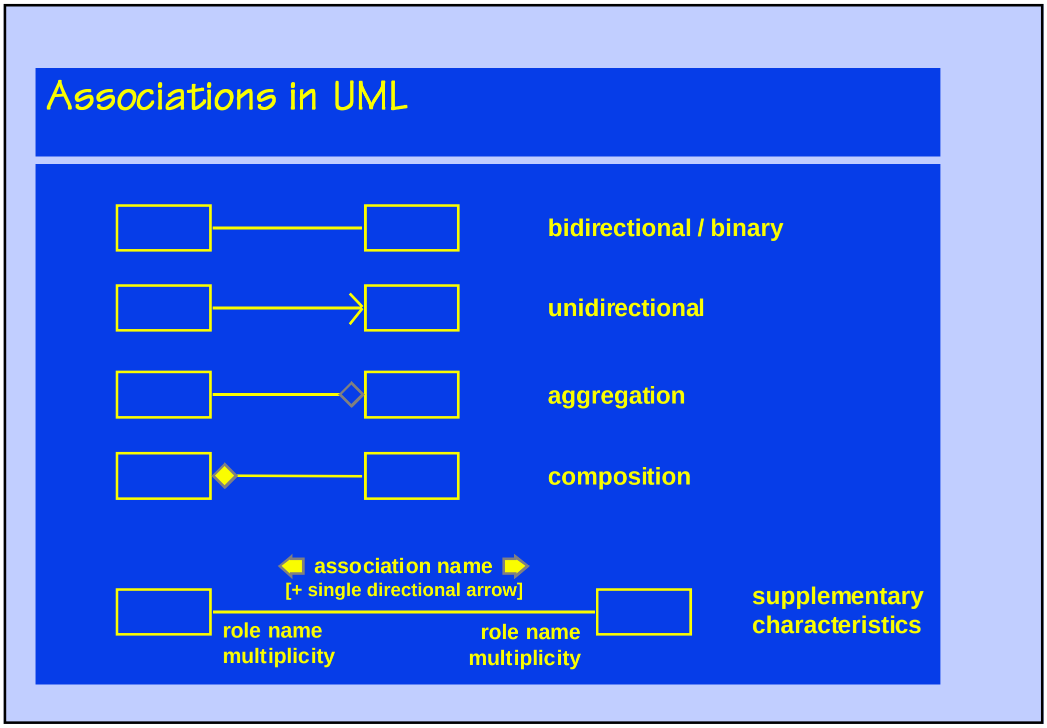
SUBCLASS

Handler

KeyboardHandler

MouseHandler

JoystickHandler



Associations in UML

bidirectional / binary

unidirectional

aggregation

composition

association name

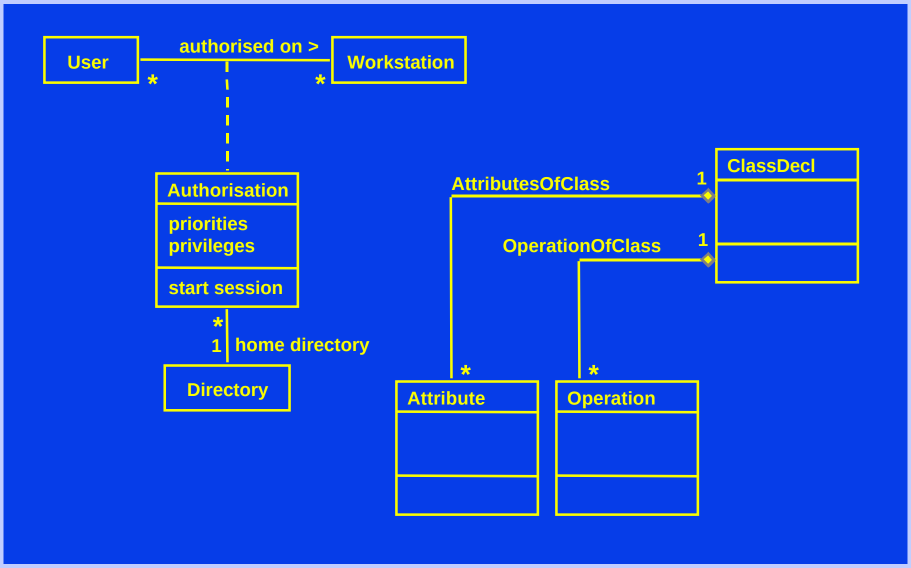
[+ single directional arrow]

role name
multiplicity

role name
multiplicity

supplementary
characteristics

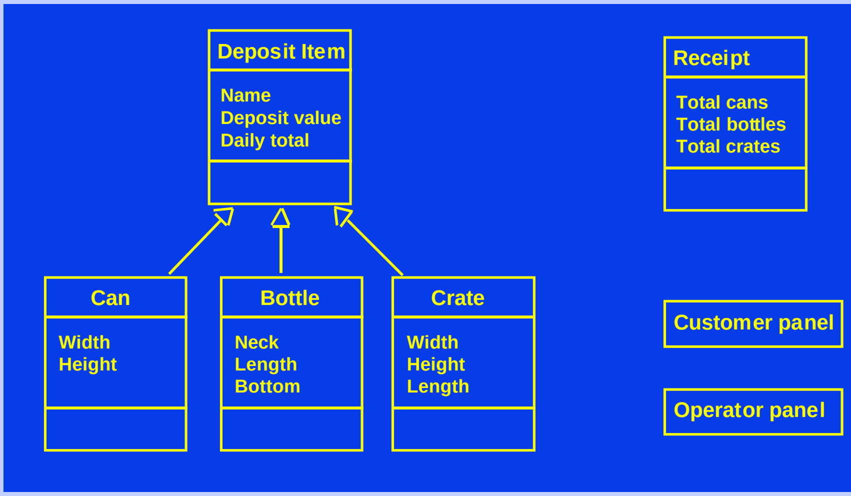
Association Examples in UML



Class Diagram in UML

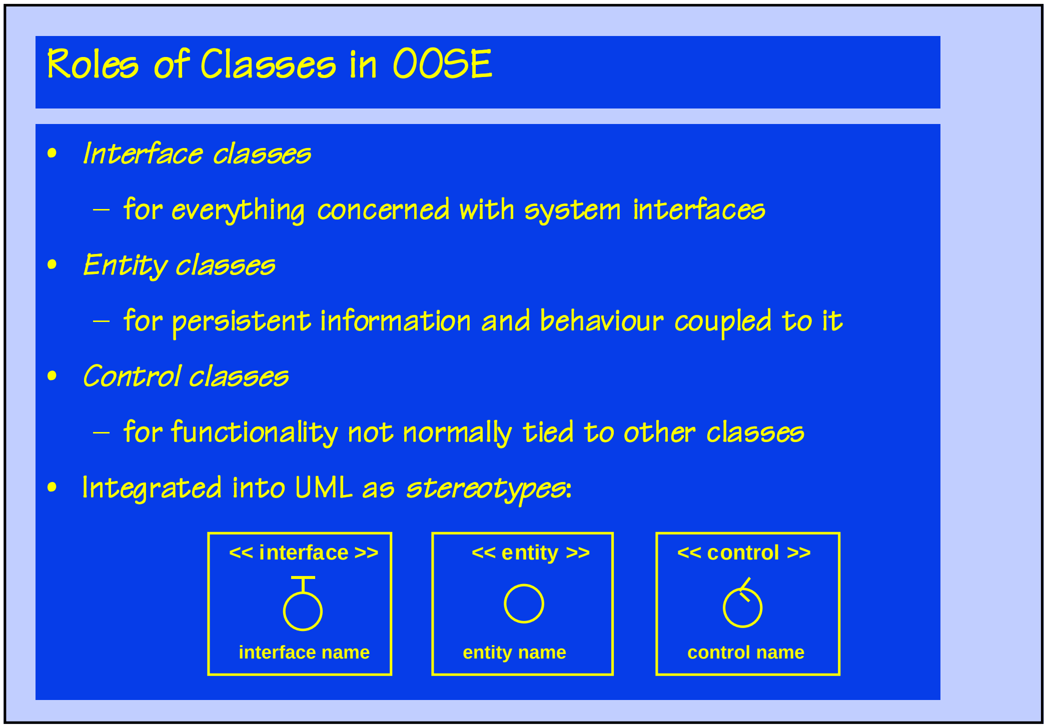
- *Class diagrams*
  - show logical, static structure of system
  - provide core of 'unified model'
- *Generation of initial class diagram* from *problem domain object list*
  - classes of objects
  - associations / attributes
  - inheritance relationships

Initial Class Diagram for Recycling Machine



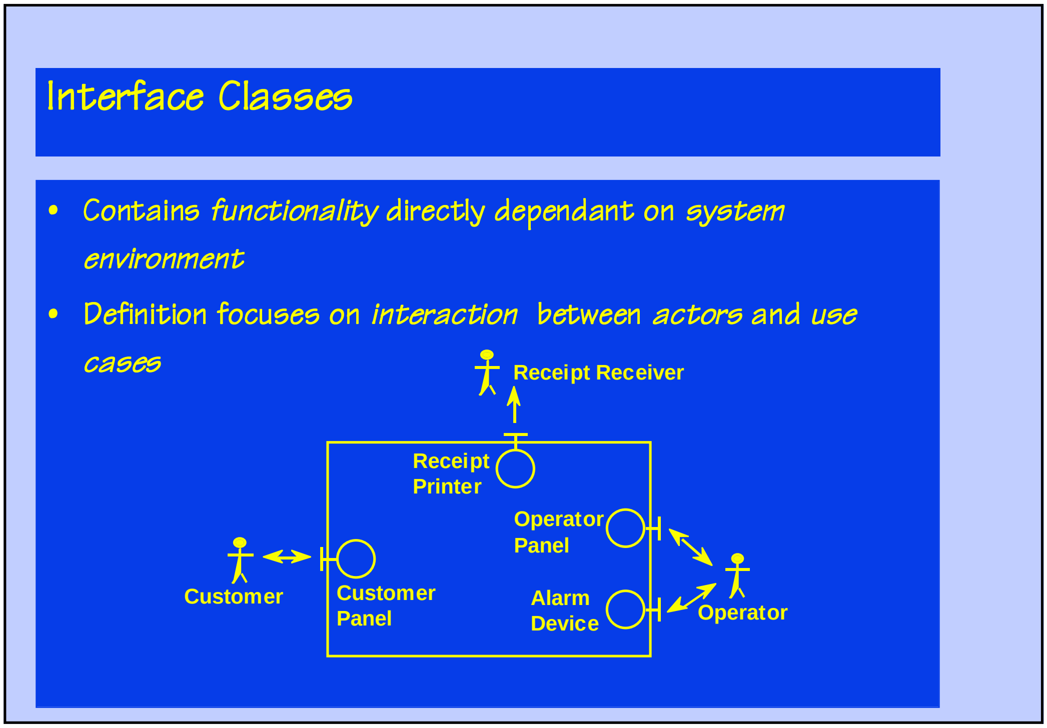
Exploiting Use Cases

- Employ *classes* and *use cases*, one by one
  - to describe *roles* and *responsibilities* of each class
  - to distribute *behaviour* specified in *use cases*
  - to ensure that there is a *class* for *every behaviour*



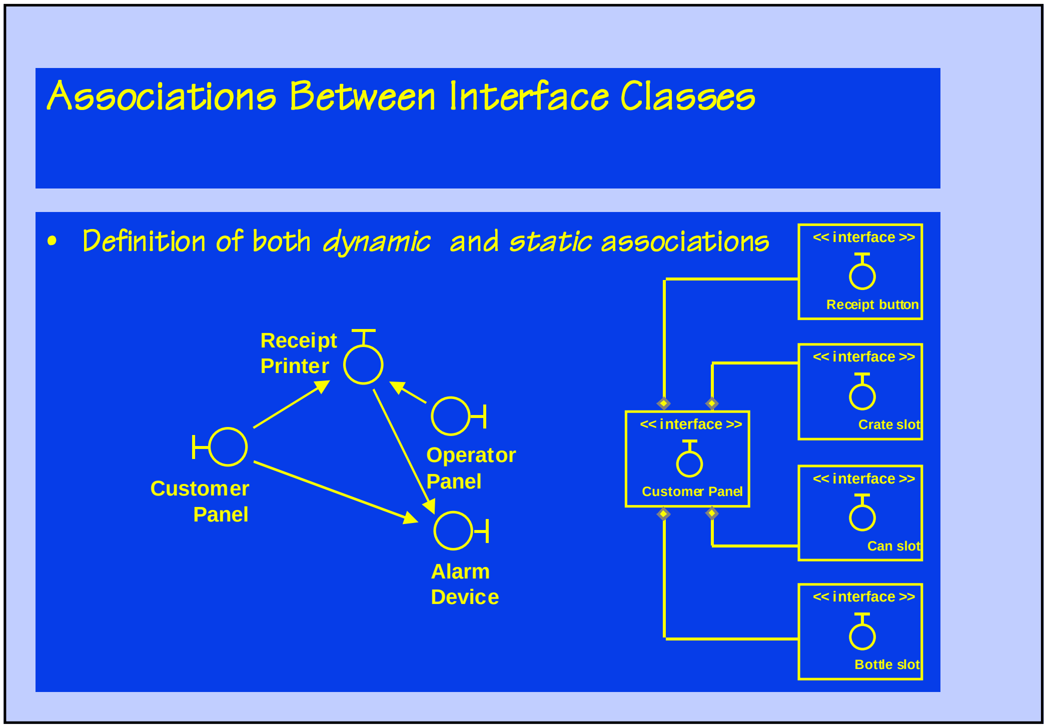
Roles of Classes in OOSE

- *Interface classes*
  - for everything concerned with system interfaces
- *Entity classes*
  - for persistent information and behaviour coupled to it
- *Control classes*
  - for functionality not normally tied to other classes
- Integrated into UML as *stereotypes*:



Interface Classes

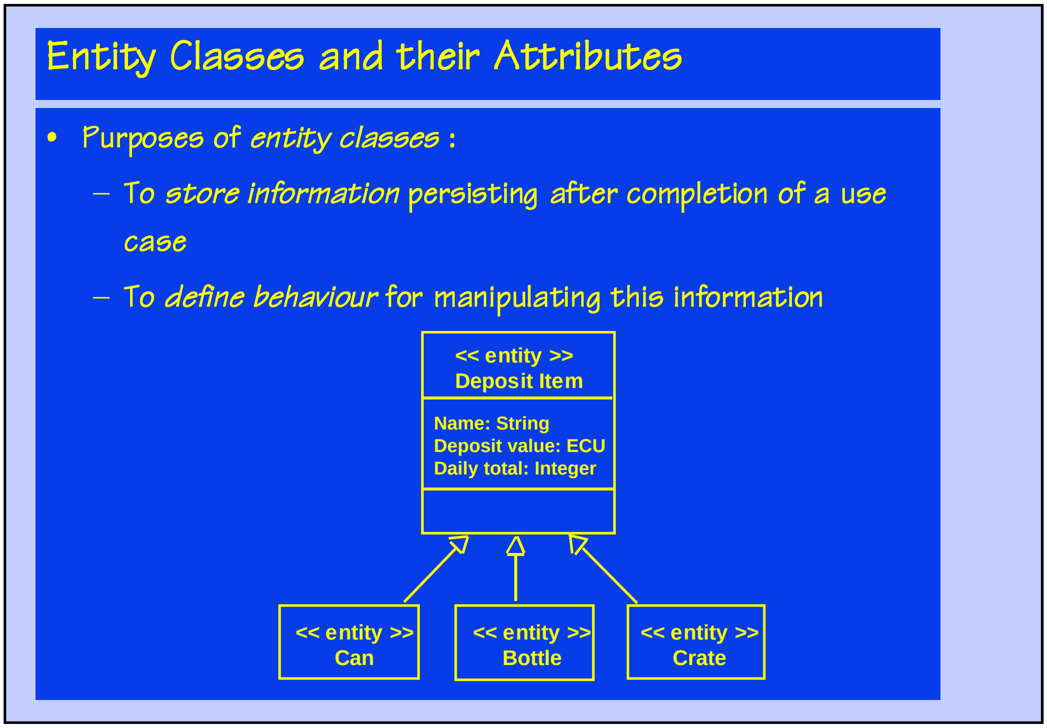
- Contains *functionality* directly dependant on *system environment*
- Definition focuses on *interaction* between *actors* and *use cases*



Associations Between Interface Classes

- Definition of both *dynamic* and *static associations*

Receipt Printer
Customer Panel
Operator Panel
Alarm Device
<< interface >> Receipt button
<< interface >> Crate slot
<< interface >> Customer Panel
<< interface >> Can slot
<< interface >> Bottle slot

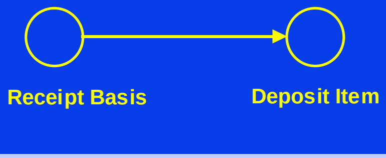


Entity Classes and their Attributes

- Purposes of *entity classes* :
  - To *store information* persisting after completion of a use case
  - To *define behaviour* for manipulating this information

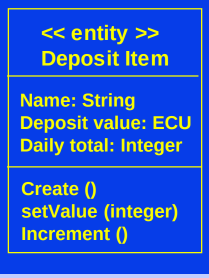
Entity Communication

- A primary task to identify associations involving communication
  - modelling of *communication* between objects
  - shows the sending and receiving of *messages* as stimuli
  - *starts from* object initiating communication
  - *directed to* object where reply generated or operation executed



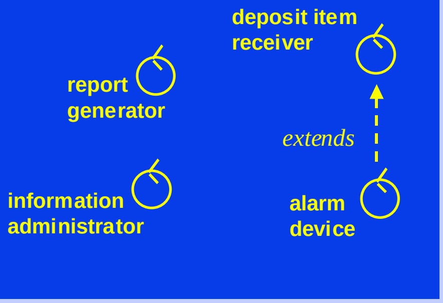
Entity Operations

- Defining *entity operations* for:
  - *storing* and *fetching* information
  - *creating* and *removing* object
  - behaviour that must change if entity object is changed



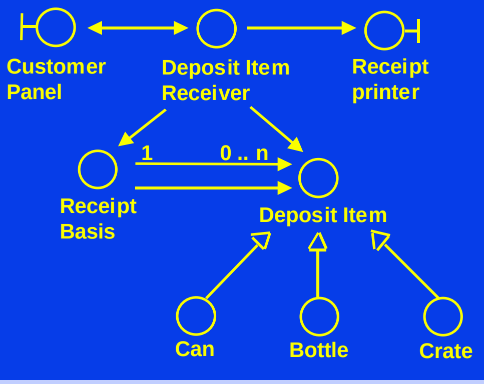
Control Classes

- *Control classes* needed to provide for:
  - behaviour not natural in *interface* and *entity* classes
  - '*glue*' between other classes in use case
  - *typical control behaviours*
  - improved maintainability



Use Case View

- *Model* each use case
- *Describe* use case in terms of *classes*



An Elaborated Use Case

- When the customer returns a deposit item the *Customer Panel's* sensors measure its dimensions. These measurements are sent to the control object *Deposit Item Receiver* which checks via *Deposit Item* whether it is acceptable. If so, *Receipt Basis* increments the customer total and the daily total is also incremented. If it is not accepted, *Deposit Item Receiver* signals this back to *Customer Panel* which signals NOT VALID.
- When the Customer presses the receipt button, *Customer Panel* detects this and sends this message to *Deposit Item Receiver*. *Deposit Item Receiver* first prints the date via *Receipt Printer* and then asks *Receipt Basis* to go through the customer's returned items and sum them. This information is sent back to *Deposit Item Receiver* which asks *Receipt Printer* to print it out.

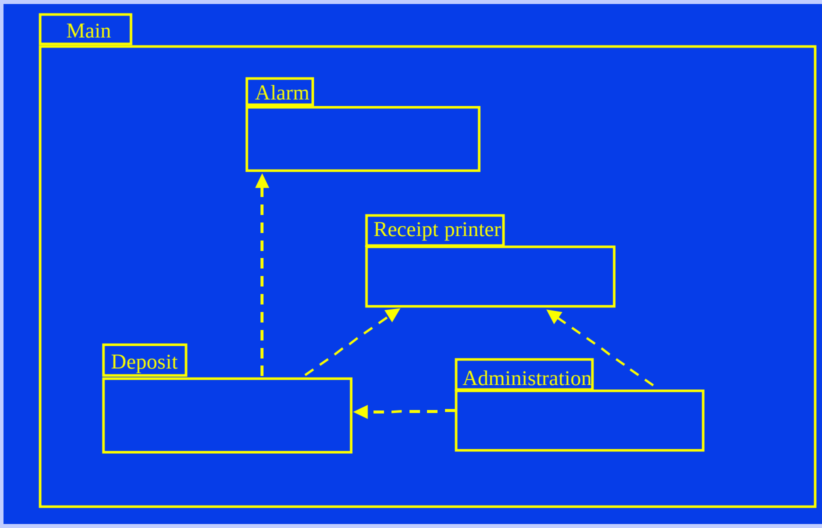
Packages

- Packages are necessary:
  - because of *large numbers of classes*
  - to provide *optional functionality*
  - to minimise *effect of change*
- Packages should have a:
  - *tight* functional *coupling inside*
  - *weak coupling outside* indicated by 'dependency associations' between packages

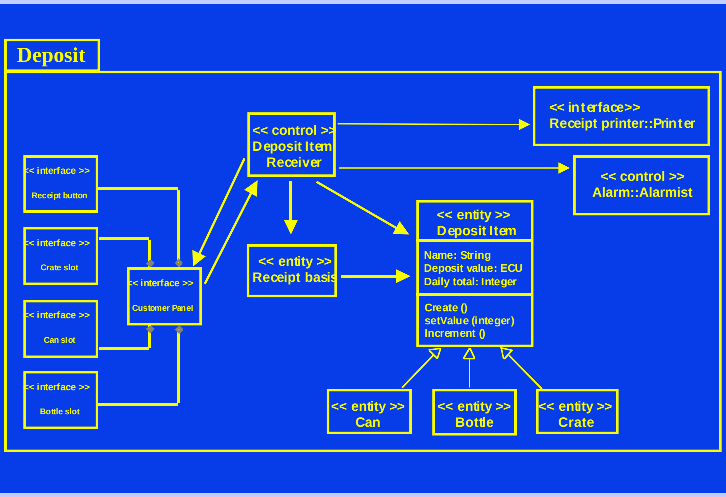
Packages (Continued)

- Packages may:
  - '*contain*' nested packages with 'service packages' as atomic parts
  - have individual *classes outside*
  - be result of organisational or managerial *pressures*

Recycling Machine Packages

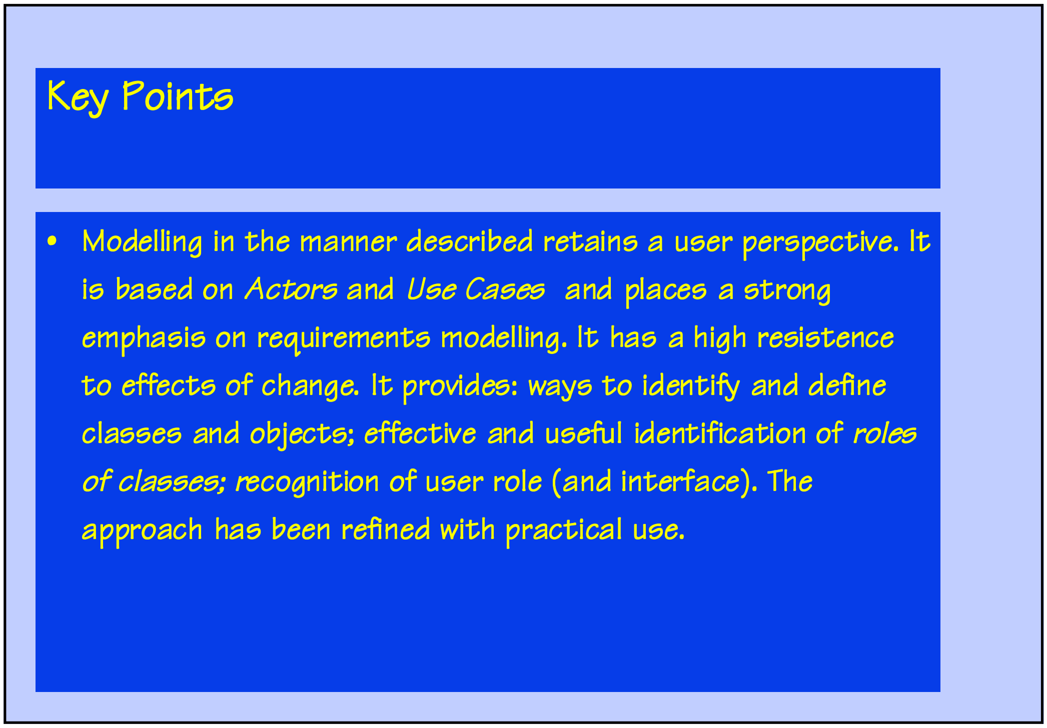


“Deposit” Package in UML



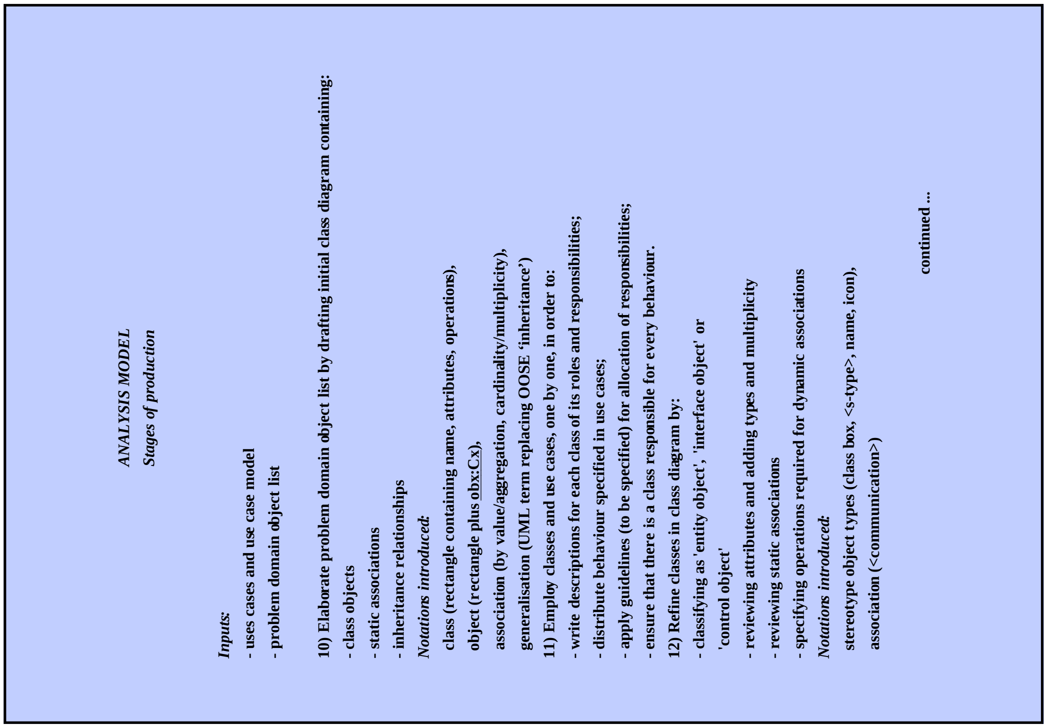
Analysis Model

- Outputs:
  - *class roles* [text]
  - *use case description in terms of classes and operations* [text x use case]
  - *completed analysis model classes* [diagram]
  - *sub-system diagrams* [package diagram]
- Notations introduced:
  - *class, object, associations* (binary, unidirectional, aggregation, generalisation)
  - stereotypes (classes, associations)
  - package (+ dependancy association)



Key Points

- Modelling in the manner described retains a user perspective. It is based on *Actors* and *Use Cases* and places a strong emphasis on requirements modelling. It has a high resistence to effects of change. It provides: ways to identify and define classes and objects; effective and useful identification of *roles of classes; recognition of user role (and interface).* The approach has been refined with practical use.



ANALYSIS MODEL

Stages of production

Inputs:

- uses cases and use case model

- problem domain object list

10) Elaborate problem domain object list by drafting initial class diagram containing:

- class objects

- static associations

- inheritance relationships

Notations introduced:

class (rectangle containing name, attributes, operations),

object (rectangle plus obx:Cx),

association (by value/aggregation, cardinality/multiplicity),

generalisation (UML term replacing OOSE 'inheritance')

11) Employ classes and use cases, one by one, in order to:

- write descriptions for each class of its roles and responsibilities;

- distribute behaviour specified in use cases;

- apply guidelines (to be specified) for allocation of responsibilities;

- ensure that there is a class responsible for every behaviour.

12) Refine classes in class diagram by:

- classifying as 'entity object', 'interface object' or 'control object'

- reviewing attributes and adding types and multiplicity

- reviewing static associations

- specifying operations required for dynamic associations

Notations introduced:

stereotype object types (class box, <s-type>, name, icon),

association (<communication>)

continued ...

ANALYSIS MODEL

Stages of production (continued)

13) Execute check by:

- rewriting textual descriptions of use case in terms of classes and atomic operations.

14) Revise class diagram

15) Group objects into:

- atomic <service packages>

- larger <sub-systems> and their dependent packages

Notations introduced:

package

dependancy association

Outputs:

- object roles and responsibilities [text],

- use case description in terms of objects and operations [text x use case],

- completed analysis model class diagram,

- sub-system diagrams [package diagram]

Stereotype icons for use after, rather than before, class definition.