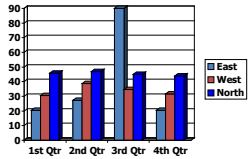
Object-Oriented Programming

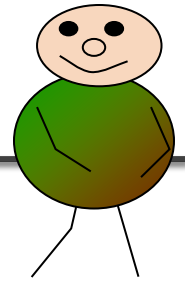
Yunho Kim
Hanyang Univ.

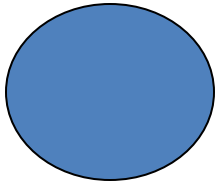
What is an Object?

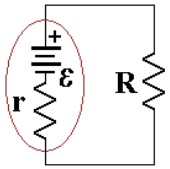
- An object is an entity which consists of a number of operations and a state which remembers the effect of the operations.
- State is regarded as information which is saved in a set of variables.
- Operations (aka. methods) read and/or affect the information.
- The only part that can be seen by an outsider are the operations. For this reason, operations are also known as behaviours.
- Information is hidden to the outsider.

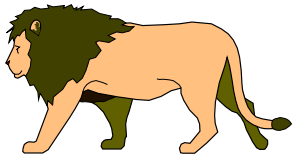
What is an Object?







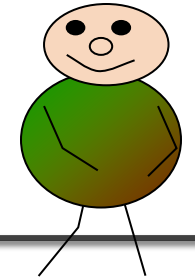




- Tangible Things as a car, printer, ...
- Roles as employee, boss, ...
- Incidents as flight, overflow, ...
- Interactions as contract, sale, ...
- Specifications as colour, shape, …

- An object represents an individual, identifiable item, unit, or entity, either real or abstract, with a well-defined role in the problem domain.
- An "object" is anything to which a concept applies.

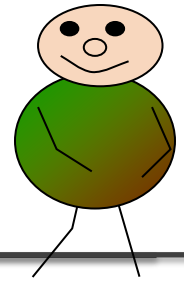
Example:
A Person as an Object



Information:
Name,
Age,
Address,
Friends,
...

Behaviour:
jump
    bend legs
    stretch legs
    lift arms,
    ...
walk
talk

UML diagram of the Person class



Person
-name: string -yearOfBirth: int
+create(theName: string ,theYear: int) +jump() +getAge(this year: int):int …

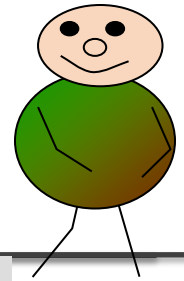
Name

Private data (attributes)

Initialisation (constructor)

Public behaviour (methods)

Java code of the Person class



```
public class Person {

        private String name;
        private int yearOfBirth;

        public Person(String theName, int theYear) {
            name = theName;
            yearOfBirth = theYear;
            }

        public void jump() {
            System.out.println(name + " jumps");
            }

        public int getAge(int thisYear) {
           return thisYear - yearOfBirth;
           }
}
```

Name

Private data (attributes)

Initialisation (constructor)

Public behaviour (methods)

Basic Terminology: Classes and Instances

- A *Class* is a template or abstract description of the same type of objects.
  - It defines the common features of the objects, that is the operations and information structure.
- An *Instance* is an object created from a class.
- An instance's *behaviour* and *information structure* is defined in the class.
- Its current *state* (values of instance variables) is determined by operations performed on it.

Basic Terminology: Inheritance

- https://en.wikipedia.org/wiki/Inheritance_(object-oriented_programming)

  inheritance is the mechanism of basing an object or class upon another object (prototype-based inheritance) or class (class-based inheritance), retaining similar implementation

- The basic principle is simple:

  A class gets the state and behaviour of another class and adds additional state and behaviour.

What is the purpose of Inheritance?

- ✧ Specialisation:
  - Extending the functionality of an existing class.
- ✧ Generalisation:
  - Sharing commonality between two or more classes.
- ✧ Polymorphism.
  - Implementation of different behaviour to the same message (depending on the type of the object).

Inheritance - related Terminology

- ✧ Derived class or subclass or child class.
  - A class which inherits some of its attributes and methods from another class.
- ✧ Base class or superclass or parent class.
  - A class from which another class inherits.
- ✧ Ancestor.
  - A class's ancestors are those from which its own superclasses inherit.
- ✧ Descendant.
  - A class's descendants are those which inherit from its subclasses.

Inheritance vs. Aggregation

- Inheritance means that one class inherits the characteristics of another class.
  - This is also called a “is a” relationship:

A car *is a* vehicle

A student *is a* person

- Aggregation describes a “has *a*” relationship.
  - One object is a part of another object.

A car *has* wheels

A person *has* legs

Basic Terminology: Polymorphism

- ✧ Original meaning: “Having many forms”
- ✧ The sender of a stimulus doesn’t need to know the receiver’s class.
- ✧ Different receivers can interpret the message in their own way.
- ✧ Consequence: different receivers can response to the same stimulus based on their interpretation.
  - So we can have a variety of objects who process the same piece of data in different ways.
- ✧ Implemented via method overloading and overriding.

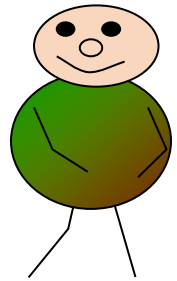
Basic Terminology: Encapsulation

- **Encapsulation** is the process of **hiding the implementation details** of an object.
- The only access to manipulate the object data is through its interface.
- It protects an object's internal state from being corrupted by other objects.
- Also, other objects are protected from changes in the object implementation.
- Encapsulation allows objects to be viewed as 'black boxes'.
- Communication is achieved through an 'interface'.

Example: Encapsulation

- *Encapsulation* is the practice of including in an object everything it needs hidden from other objects.
  - The internal state is usually not accessible by other objects.

Public:
Has an age and can jump.



Private:
Has a name and a year of birth.

The Analysis Phase

- In software engineering, analysis is the process of converting the user requirements to system specification (system means the software to be developed).
- System specification, also known as the logic structure, is the developer's view of the system.
- Function-oriented analysis – concentrating on the decomposition of complex functions to simply ones.
- Object-oriented analysis – identifying objects and the relationship between objects.

Object Oriented Analysis

- Identifying objects: Using concepts, CRC cards, stereotypes, etc.
- Organising the objects: classifying the objects identified, so similar objects can later be defined in the same class.
- Identifying relationships between objects: this helps to determine inputs and outputs of an object.
- Defining operations of the objects: the way of processing data within an object.
- Defining objects internally: information held within the objects.

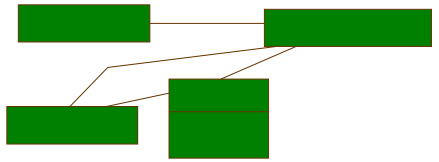
Three ways to do Object Oriented Analysis (there are more…).

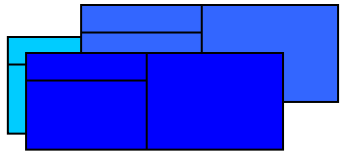
- Conceptual model (Larman)

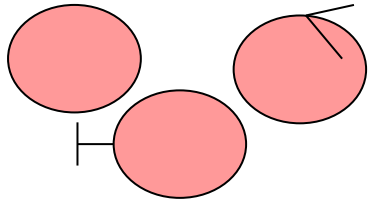
  Produce a “light” class diagram.
- CRC cards (Beck, Cunningham)

  Index cards and role playing.
- Analysis model with stereotypes (Jacobson)

  Boundaries, entities, control.







- A good analyst knows more than one strategy and even may mix strategies in order to identify the objects and relationships for the design phase.

Conceptual Model - Overview

- A conceptual model is a representation of concepts in a problem domain.
- In UML it is basically a “class diagram without operations”.
- It may show:
  - Concepts
  - Associations of concepts
  - Attributes of concepts

The conceptual model: Strategies to Identify Concepts

✧ Finding Concepts with the Concept Category List.

✧ Finding Concepts with
Noun Phrase Identification.

- A central distinction between object oriented and structures analysis: division by concepts (objects) rather than division by functions.

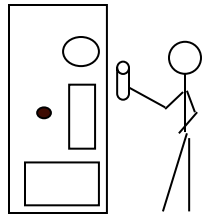
The Concept Category List

- physical or tangible objects
- specifications, designs, descriptions of things
- places
- transactions
- transaction line items
- roles of people
- containers of other things
- things in a container
- abstract noun concepts
- organisations
- events
- processes
- rules and policies
- catalogues
- records
- services
- manuals, books

Finding Concepts with Noun Phrase Identification

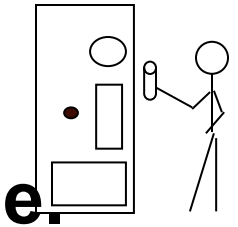
- Identify the noun and noun phrases in textual descriptions of a problem domain and consider them as candidate concepts or attributes.
- A mechanical noun-to-concept mapping isn't possible, and words in natural languages are ambiguous.

The “return item” Use case.
Exercise: Find the Nouns!



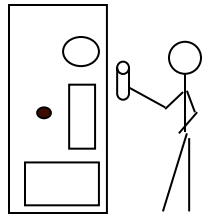
- The system controls a recycling machine for returnable bottles, cans and crates. The machine can be used by several customers at the same time and each customer can return all three types of item on the same occasion. The system has to check, for each item, what type has been returned.
- The system will register how many items each customer returns and when the customer asks for a receipt, the system will print out what was deposited , the value of the returned items and the total return sum that will be paid to the customer.
- An operator also … (not in “return item” Use Case)

Case Study: Nouns in The Recycling machine - The “return item” Use case.



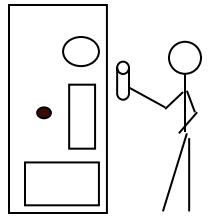
- The system controls a recycling machine for returnable bottles, cans and crates. The machine can be used by several customers at the same time and each customer can return all three types of item on the same occasion. The system has to check, for each item, what type has been returned.
- The system will register how many items each customer returns and when the customer asks for a receipt, the system will print out what was deposited , the value of the returned items and the total return sum that will be paid to the customer.
- An operator also … (not in “return item” Use Case)

Case Study:
Nouns found in the description:



- recycling machine
- bottles, cans, and crates
- machine
- customers, customer
- types of item, item, type, returned items
- system
- receipt
- return sum

Case Study:
Discussion of “recycling machine”.

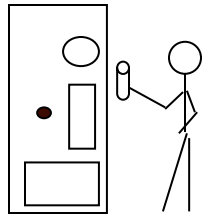


- recycling machine
- bottles, cans, and crates
- machine
- customers, customer
- types of item, item, type, returned items
- system
- receipt
- return sum

- This concept is the “overall system”. As we consider only one single use case, it is better to name this concept in the context of this use case, e.g.

  Deposit item receiver

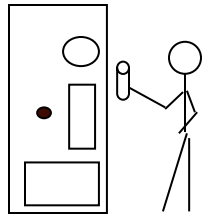
Case Study:
Discussion of “bottles, cans, and crates”.



- deposit item receiver
- bottles, cans, and crates
- machine
- customers, customer
- types of item, item, type retur
- system
- receipt
- return sum

- In a conceptual model it is usually better to use singular and multiplicities instead of plural.
- As bottle, can and crate have much in common (they are processed as items), they could be generalised to an “item”. We should remember this for later (inheritance).

Case Study:
Discussion of “machine” and “system”.

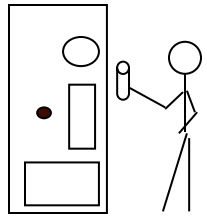


- deposit item receiver
- bottle, can, crate
- machine
- customers, customer
- types of item, item, type, returned items
- system
- receipt
- return sum

- “Machine” and “System” mean here the same, namely the “Recycling machine”, i.e. the

  Deposit item receiver

Case Study:
Discussion of “customers” and “customer”.

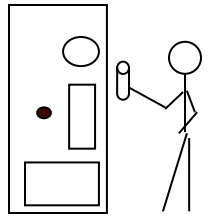


- deposit item receiver
- bottle, can, crate
- customers, customer
- types of item, item, type, returned items
- receipt
- return sum

- The customer has already been identified as an actor.
- They are outside of the system.
- We establish a concept, that interfaces with the customer (and is inside the system):

Customer panel

Case Study: Discussion of “item” (etc.).

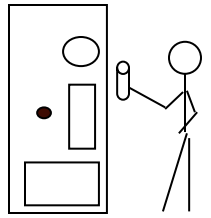


- deposit item receiver
- bottle, can, crate
- customer panel
- types of item, item, type, returned items
- receipt
- return sum

- The items that are inserted in the machine.
- Good candidate as superclass for bottle, can, crate.
- Let’s call it

  Deposit item

Case Study:
Discussion of “receipt”.



- deposit item receiver
- bottle, can, crate

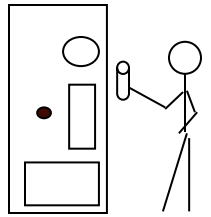
- customer panel
- deposit item

- receipt
- return sum

- The concept that “remembers” all items inserted in the machine.
- To distinguish it from the piece of paper returned to the customer, call it

  Receipt basis

Case Study:
Discussion of “return sum”.



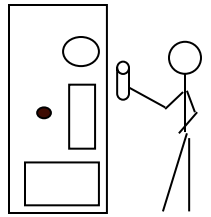
- deposit item receiver
- bottle, can, crate

- customer panel
- deposit item

- receipt basis
- return sum

- The sum that it is returned to the customer is actually computed by adding up all values of the items stored in the receipt basis.
- The sum itself is only a primitive data value, and may therefore not be considered as a concept.

Case Study:
Discussion of other concepts.



- ✧ deposit item receiver
- ✧ bottle, can, crate

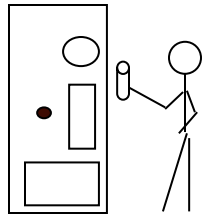
- ✧ customer panel
- ✧ deposit item

- ✧ receipt basis

- These are the concepts identified by nouns. Did we forget something?
- Check the "Concept Category List" !
- The system "interfaces" with the physical object "printer", so we add an interface concept

  Receipt printer

Case Study:
Summary of concepts identified in the analysis



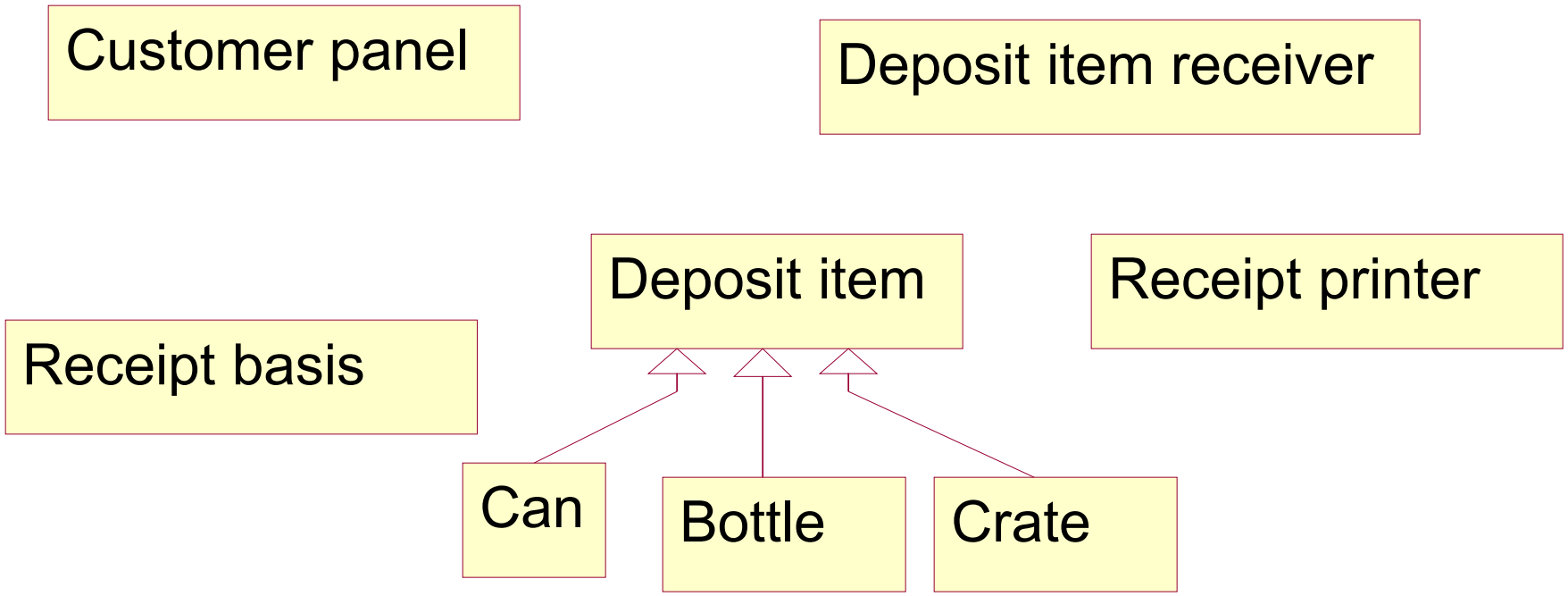
- ✧ deposit item receiver
- ✧ bottle, can, crate
- ✧ customer panel
- ✧ deposit item
- ✧ receipt basis
- ✧ receipt printer

- So far we have identified:
  - Concepts
  - A generalisation relationship.
- Next step: Finding associations.

How to make a conceptual model.

- ✧ Find the concepts
- ✧ Draw them in a conceptual model
- ✧ Add associations
- ✧ Add attributes

Drawing of Concepts



Adding Associations

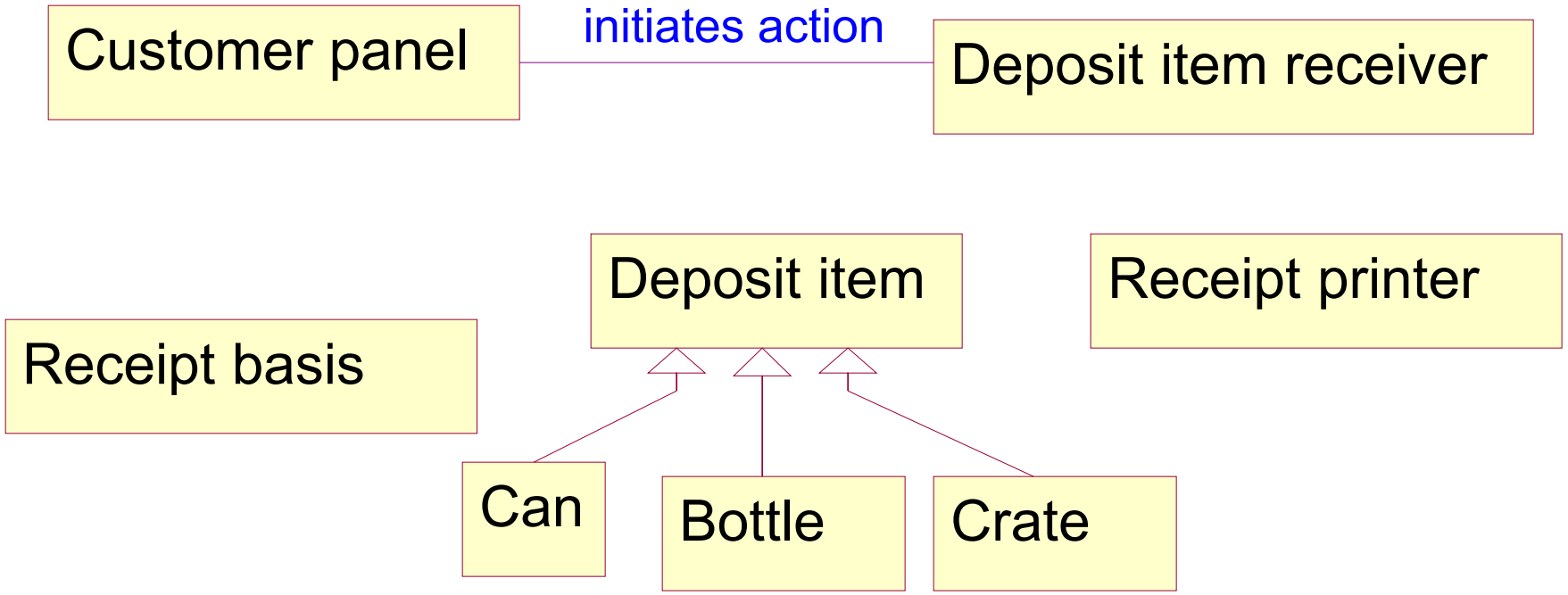
- If one concept needs to know of a another concept for some duration they should be linked by an association.
- Also use the following list in order to identify associations.

Common Association List

- A is a part of B
- A is contained in B
- A is a description for B
- A is a line item of a transaction or report B
- A is known or reported in B
- A is a member of B
- A is a organisational subunit of B
- A uses or manages B
- A communicates with B
- A is related to a transaction B
- A is a transaction related to another transaction B
- A is next to B
- A is owned by B

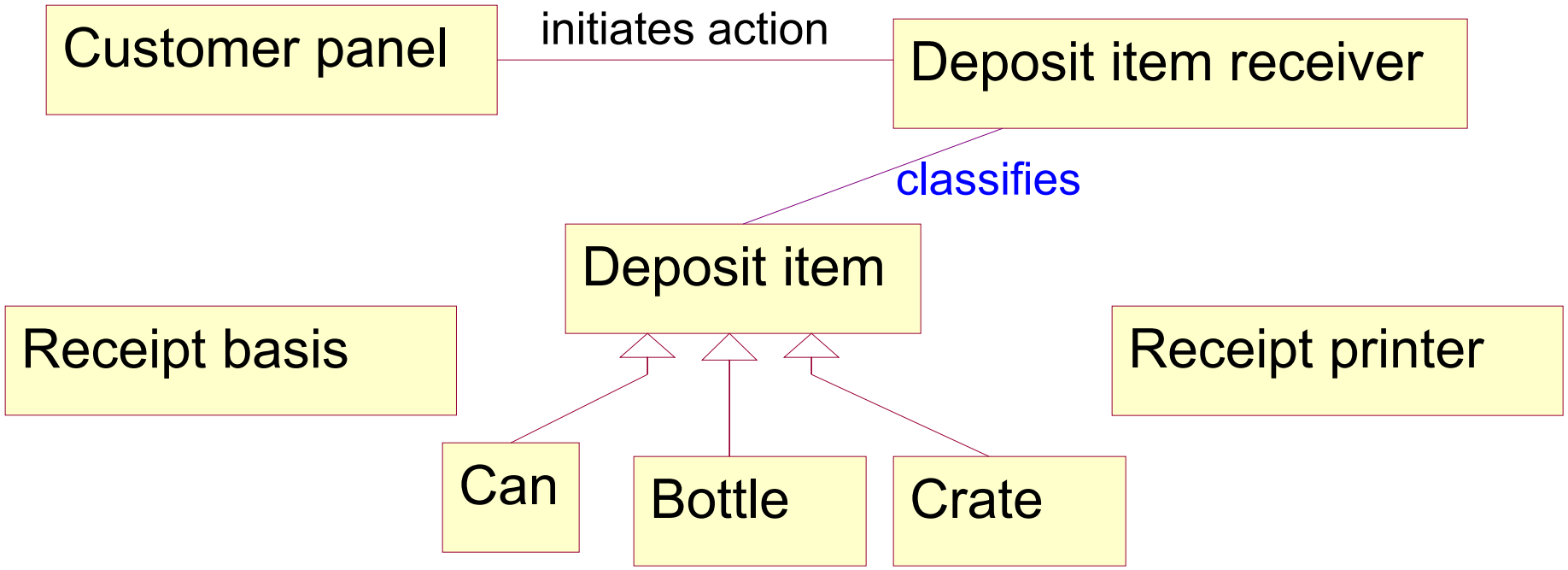
Adding associations

- The Customer panel communicates to the receiver when an item is inserted.
- Also when the receipt is requested.



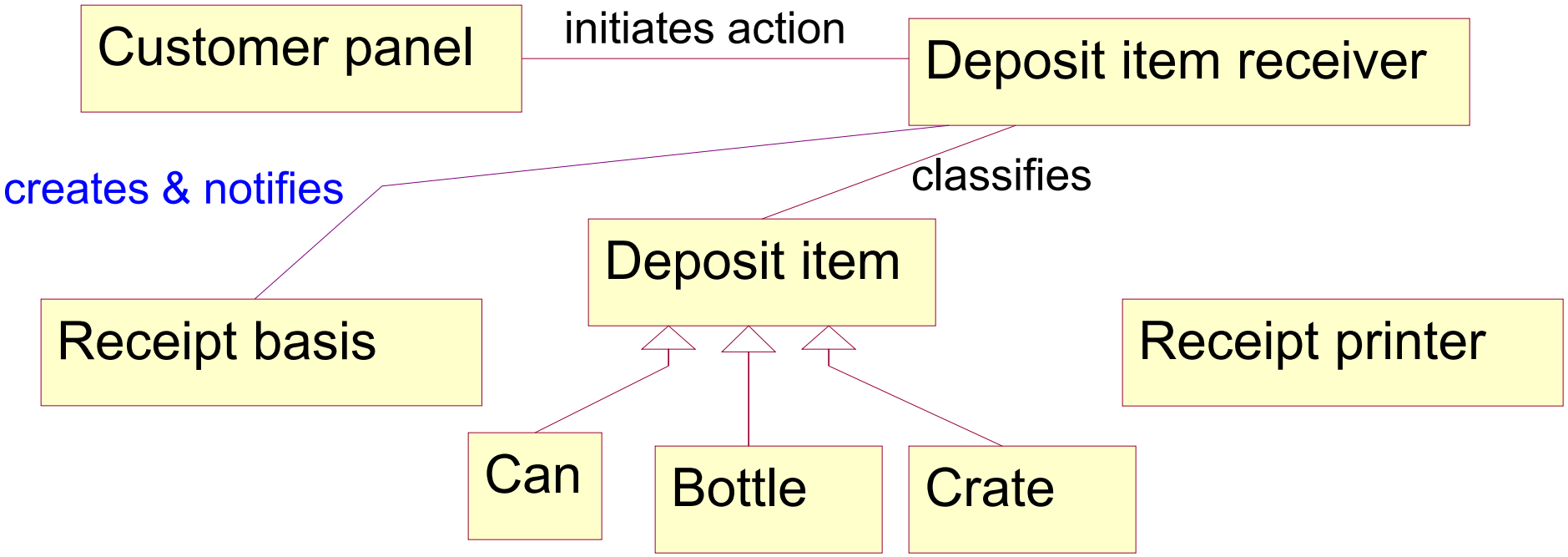
Adding associations

- The Deposit item receiver manages Deposit items:
- The items are classified.



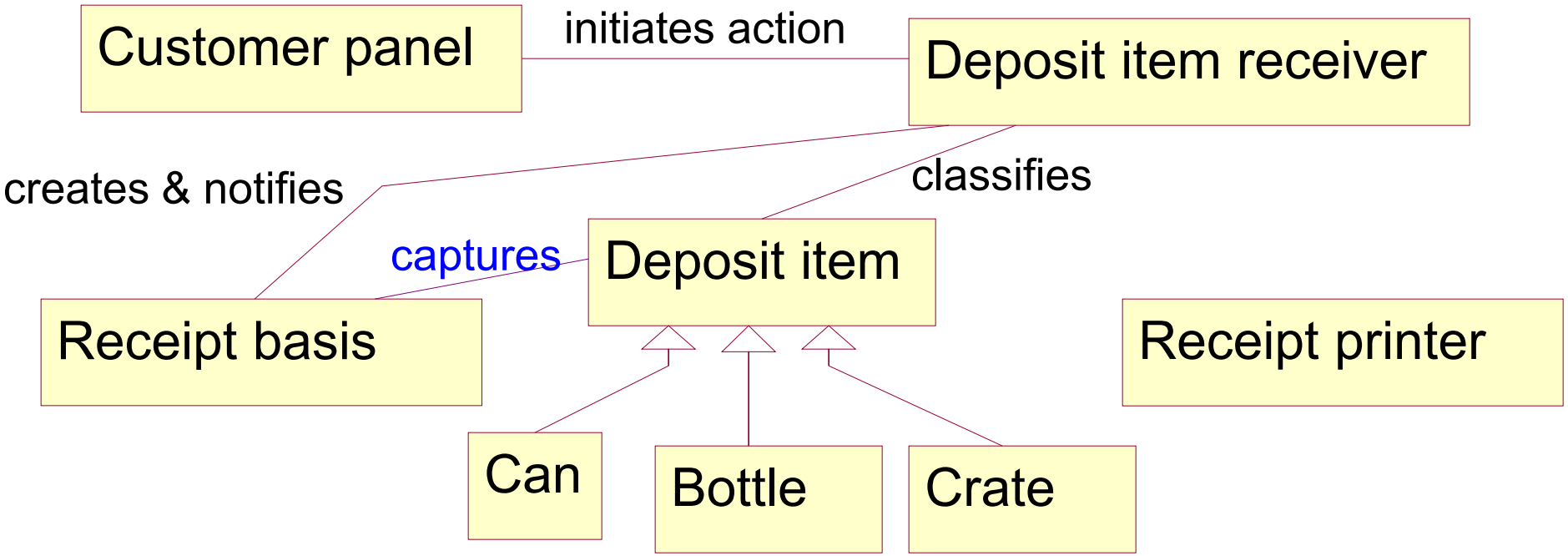
Adding associations

- The Deposit item receiver communicates to Receipt basis:
- Items received and classified are stored.
- It also creates the receipt basis when it is needed for the first time.



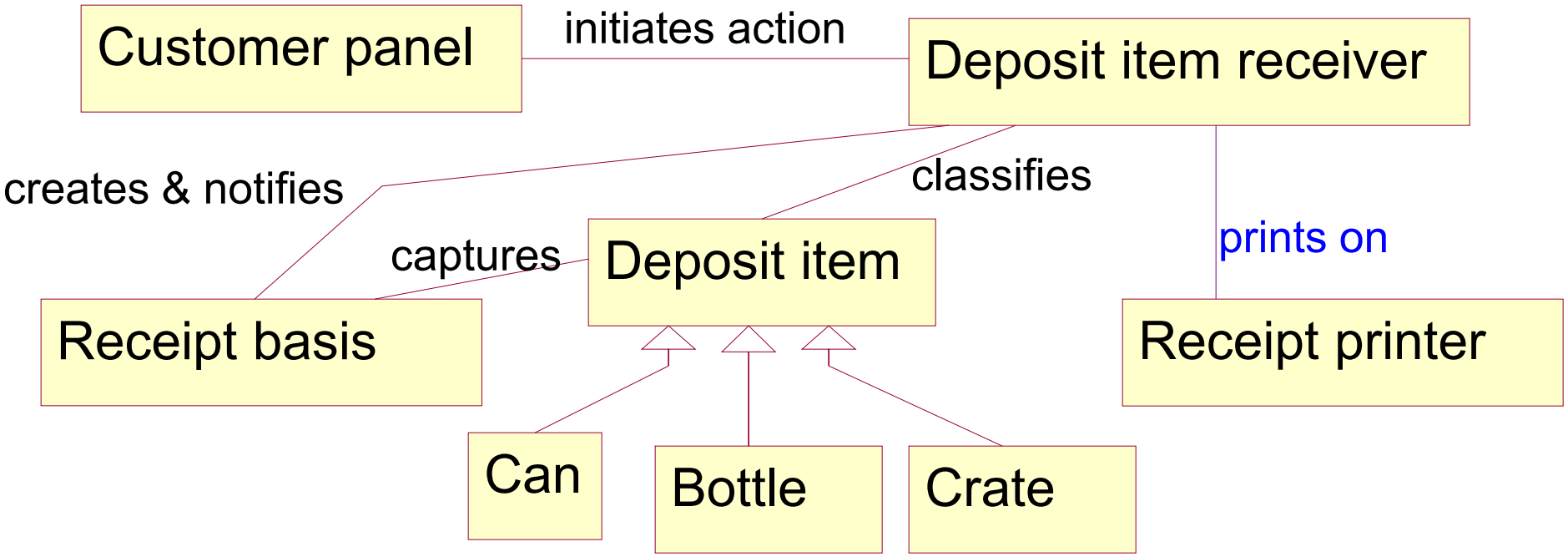
Adding associations

- The Receipt basis collects Deposit items.



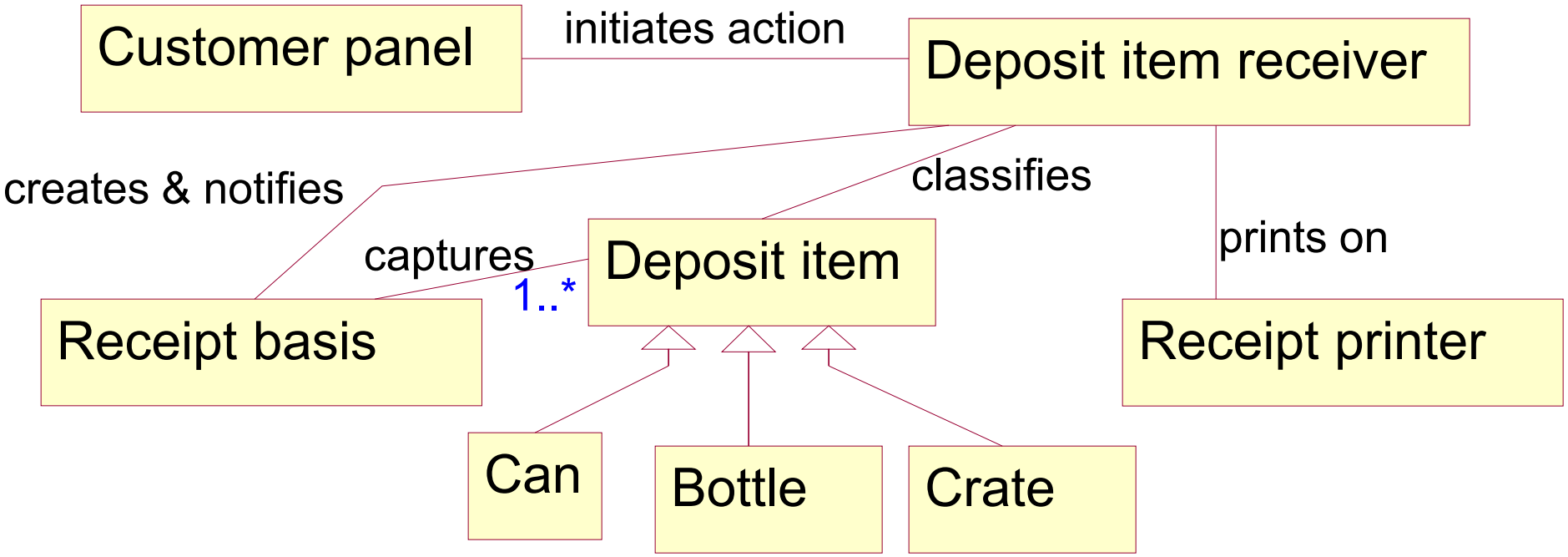
Adding associations

- On request by the Customer Panel the Deposit item receiver initiates printing of a receipt on the printer.



Adding associations

- Adding multiplicities
- Only one association here is a 1 to many relationship
- All others are 1 to 1.

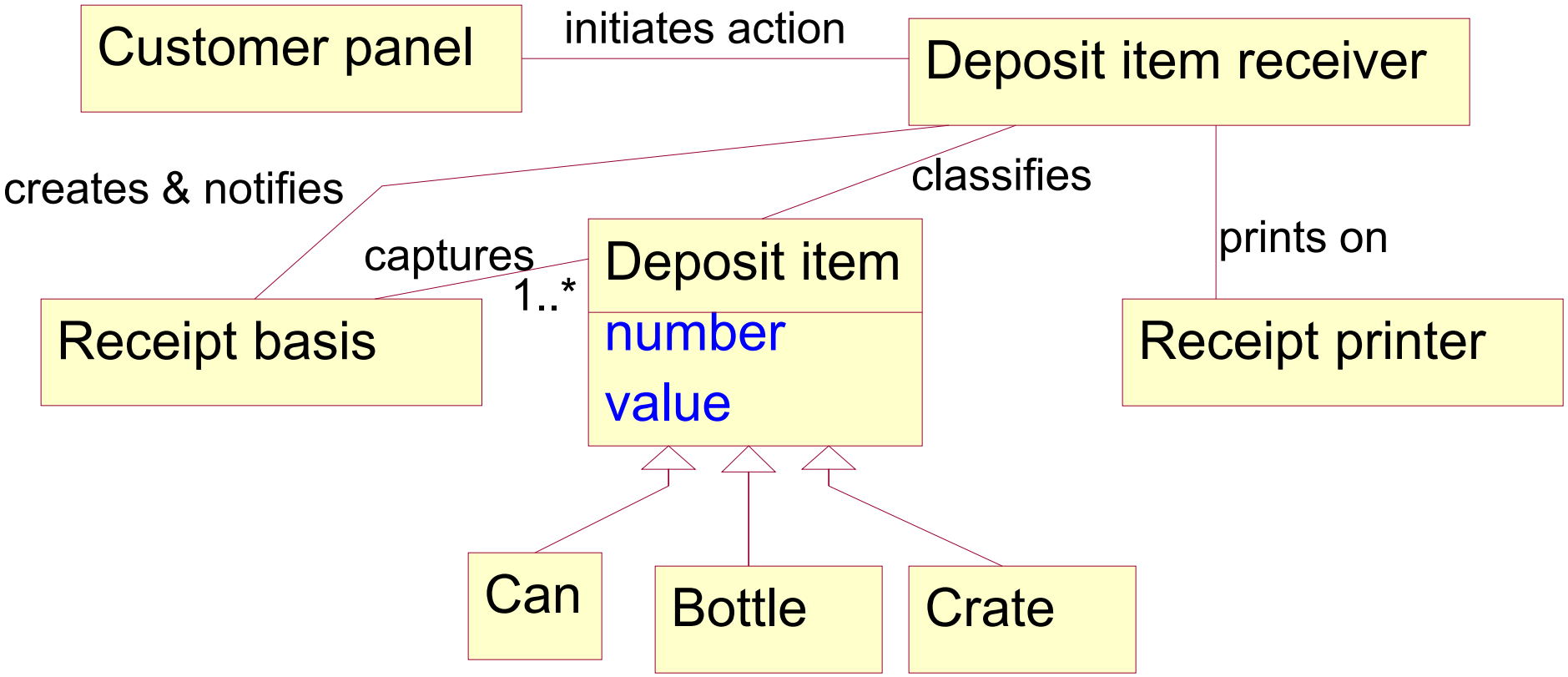


Adding Attributes

- An attribute is a logical data value of an object.
- Attributes in a conceptual model are simple data values as
  - Boolean, Date, Number, String (Text), Time, Address, Colour, Price, Phone Numbers, Product Codes, etc.
- Sometimes it is difficult to distinguish between attributed and concepts
  - E.g. Concept “Car” vs. attribute “Reg. Number”.

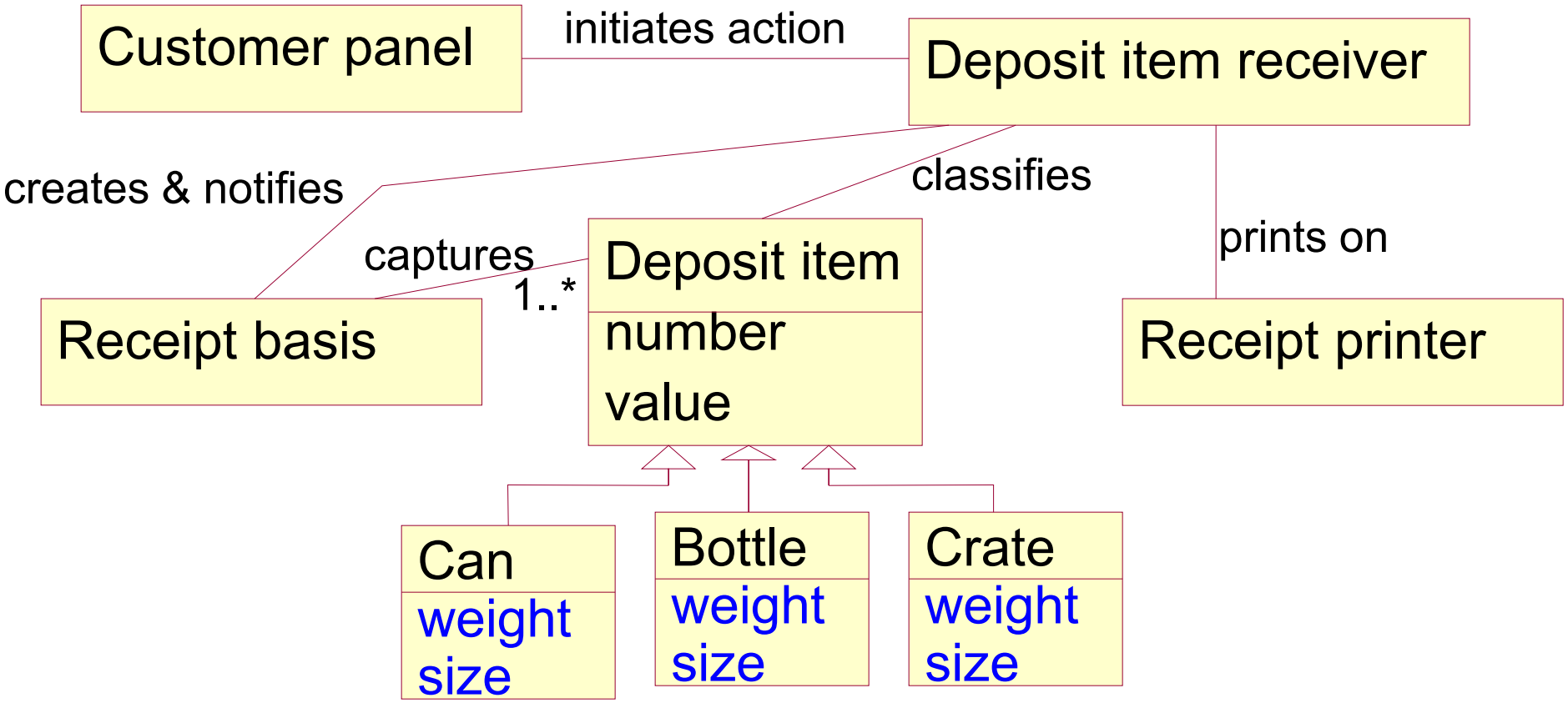
Adding attributes

- The Deposit item has a value.
- Also it will be assigned a number that shows later on the receipt.



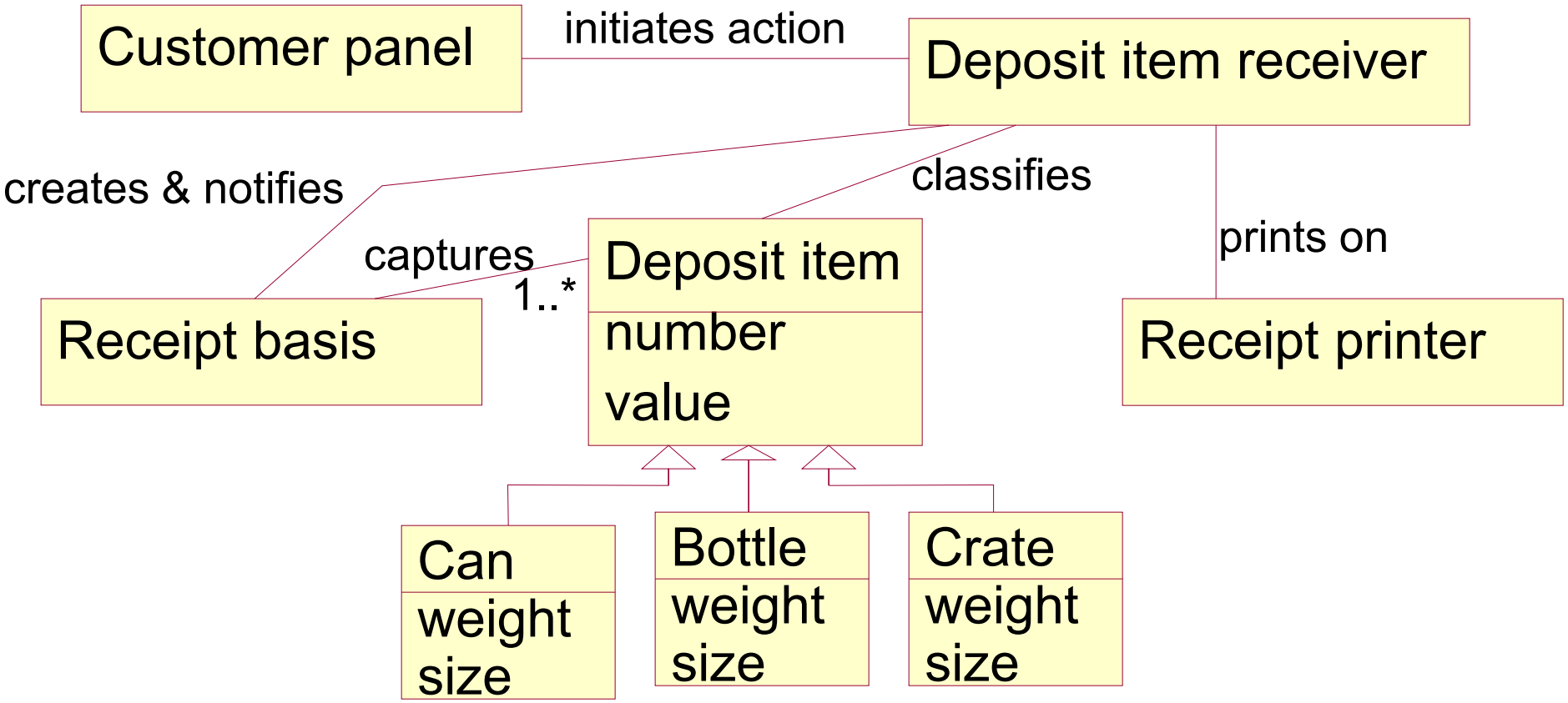
Adding attributes

- In order to be classified by the Deposit item receiver each item has also a weight and a size.
- However this is the same for each type of item, but different *between* the types.



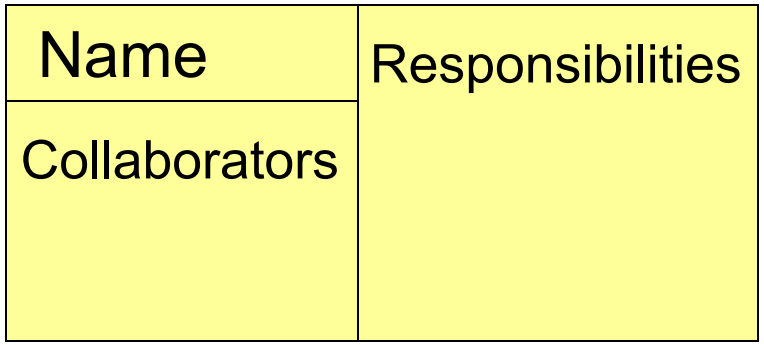
Summary

- NOT in a conceptual model:
  - arrows
  - dotted lines
  - methods, operations, functions



CRC cards & Role playing

- Not part of the UML design process but useful in detecting responsibilities of objects are CRC cards (developed by Kent Beck and Ward Cunningham).
- **CRC** stands for Class-Responsibility-Collaborator. They look like:



CRC cards & Role playing

- CRC cards are index cards, one for each class, upon which the responsibilities and collaborators of a class are written.
- They are developed in a small group session where people *role play* being the various classes.
- Each person holds onto the CRC cards for the classes that they are playing the role of.

http://www.csc.calpoly.edu/~dbutler/tutorials/winter96/crc_b/

Example: Recycling machine
CRC cards

Customer panel	▪ receive items ▪ receive print request
✧ Deposit item receiver	

Dep. item receiver	▪ classify items ▪ create Receipt Basis ▪ print receipt
▪ Printer ▪ Deposit Item ▪ Receipt Basis	

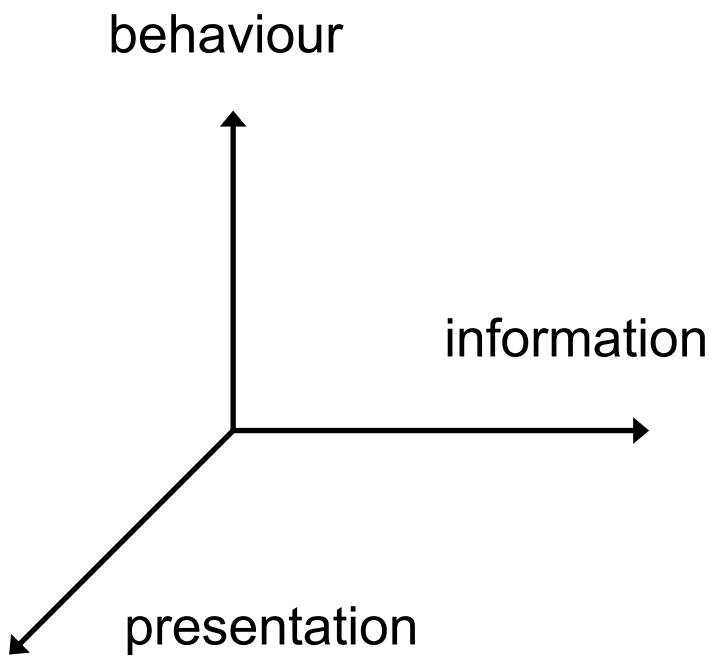
Etc.

The Object Oriented Analysis Model (Jacobson)

- ✧ An analysis model is used to represent the system specification.
- ✧ To achieve robustness and stability the model must be implementation environment independent.
- ✧ Any change in the implementation environment will not affect the logical structure of the system.
- ✧ The model must be able to capture information, behaviour (operations) and presentation (inputs and outputs).

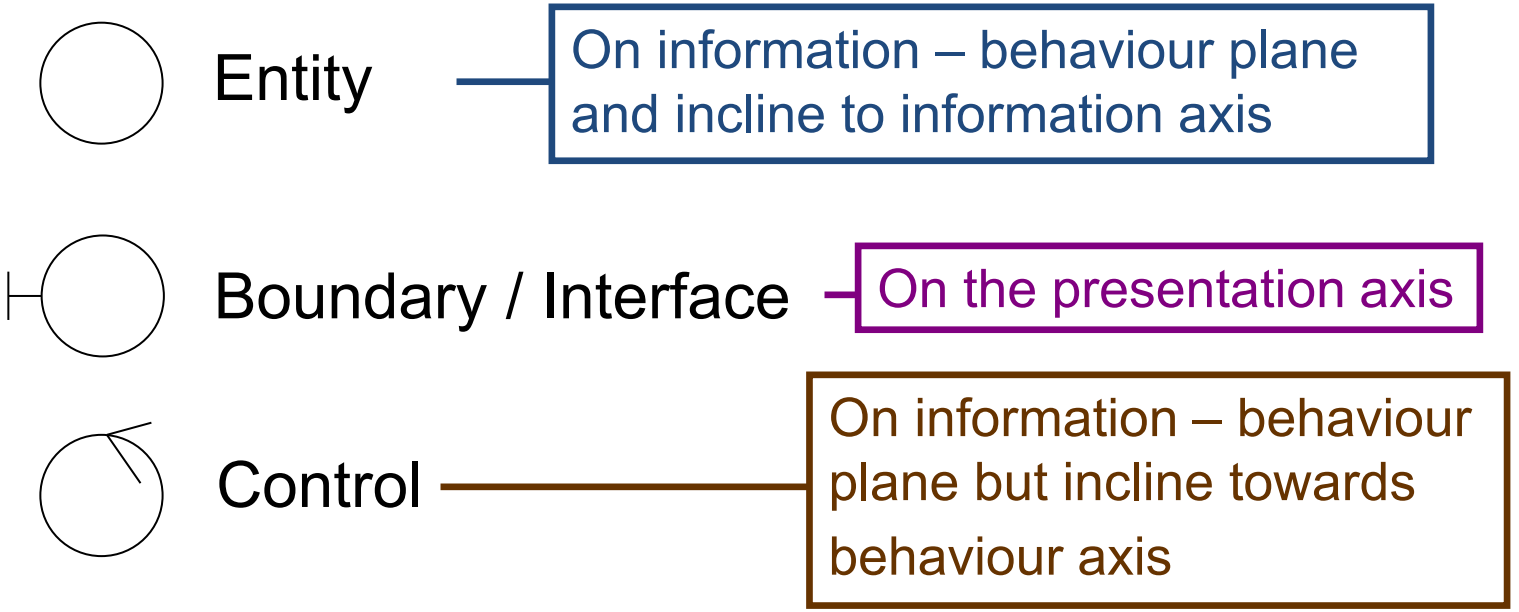
Behaviour - Information - Presentation

✧ The model is defined in information - behaviour - presentation space.

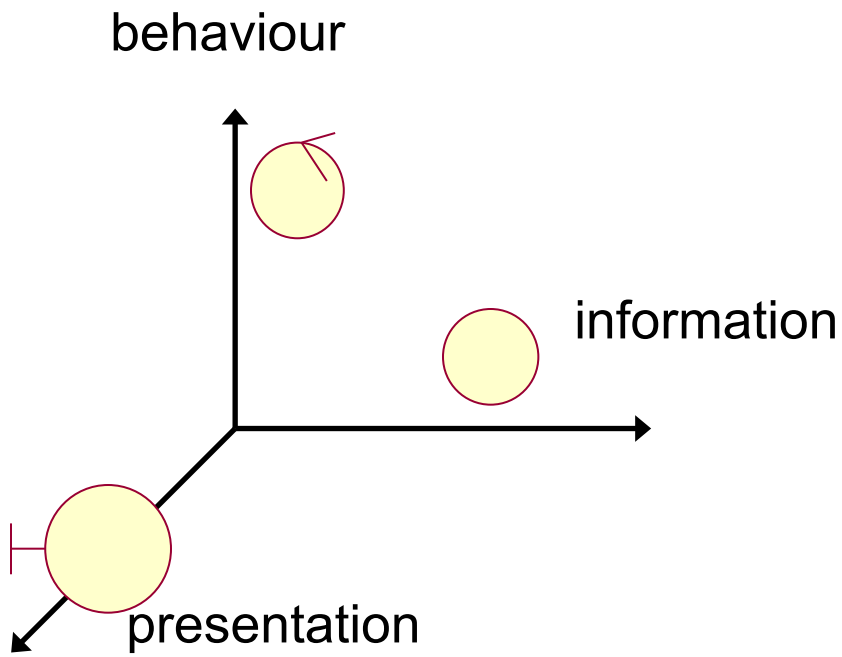


Syntax of the Object Oriented Analysis Model

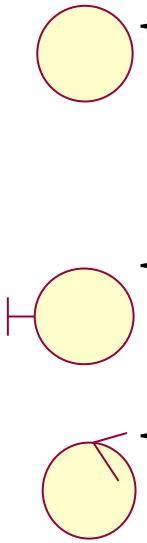
- Within an use case, we employ three types of objects (in Rational Rose, they are known as three types of entities or stereotypes):



Entity, Control, Interface



Semantics of the Object Oriented Analysis Model



- An entity object models information that shows the state of a system. This information is often used to record the effects of operations and therefore is related to the behaviours of the system.
- A boundary/interface object models inputs and outputs and operations that process them.
- A control object models functionality/operations regarding to validate and decide whether to process and pass information from the interface object to the entity object or the way around.

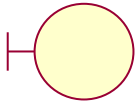
Pragmatics of the Object Oriented Analysis Model

- Identifying interface objects
  - functions directly related to actors.
- Identifying entity objects
  - information used in an use case and functions of processing the information.
- Identifying control objects
  - functions that link interface objects and entity objects

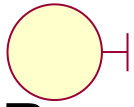
Example: The Recycling machine

- Identifying interface objects
  - Printer, Customer Panel
- Identifying entity objects
  - Long term information: Crate, Bottle, Can
  - Superclass: Deposit item
  - Short term information: Receipt basis
- Identifying control objects
  - Deposit item receiver

The Recycling machine
Interface Objects

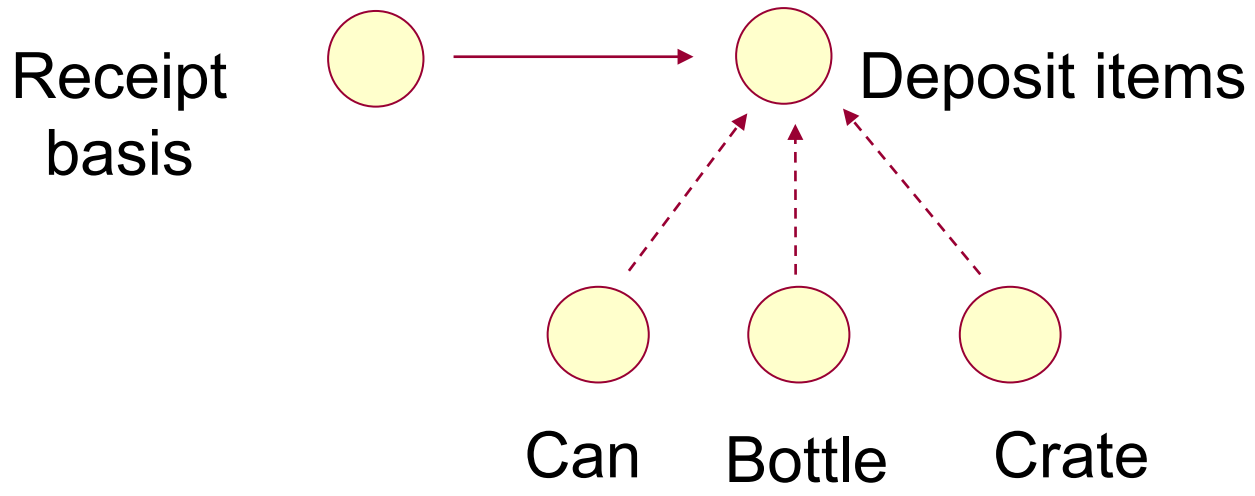


Customer panel

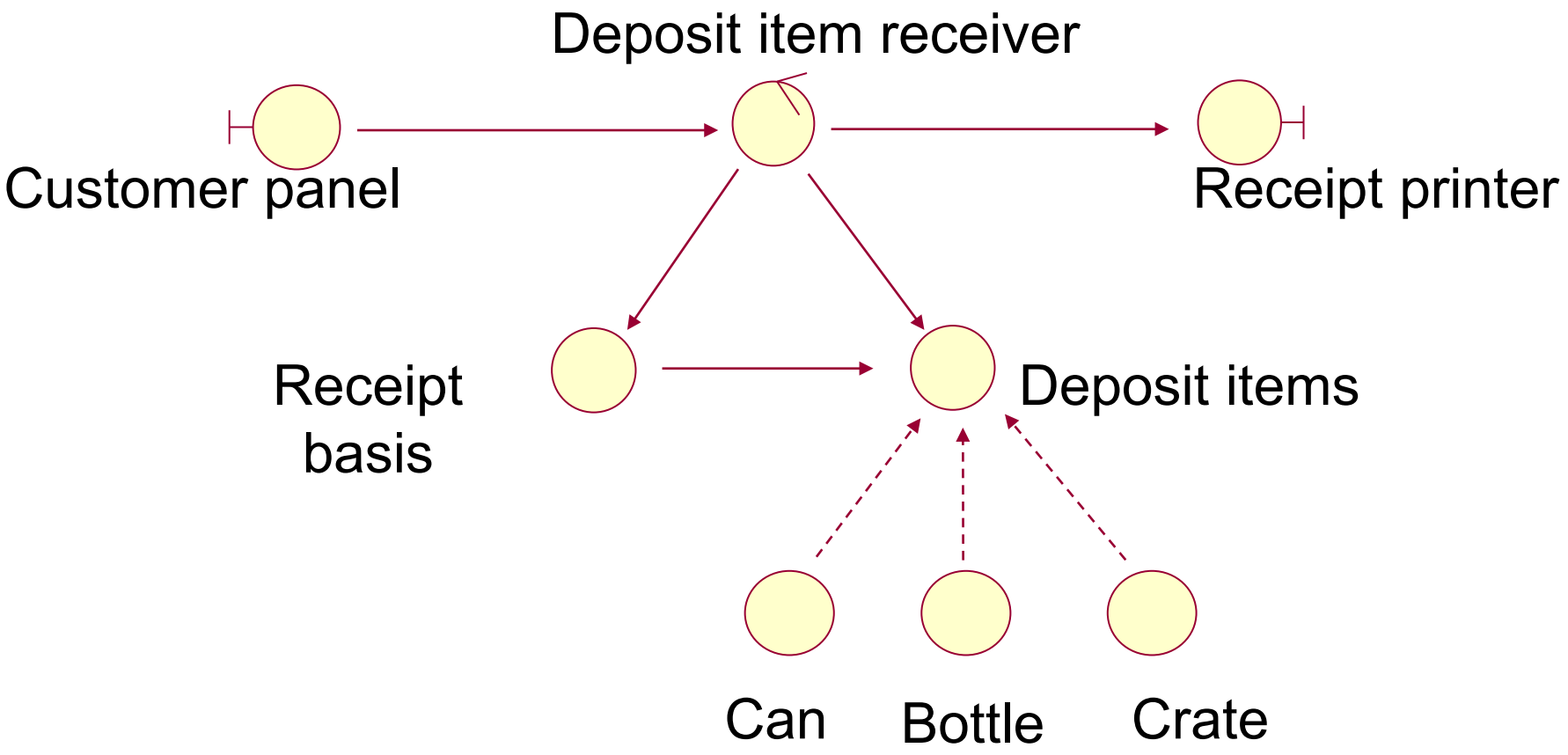


Receipt printer

The Recycling machine
Entity Objects

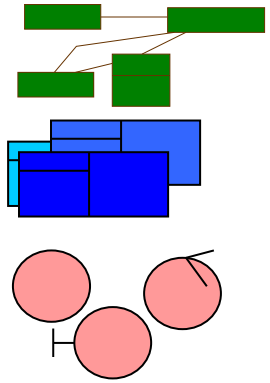


The Recycling machine
Link Interface and Entity Objects by a Control Object



Summary - Object Oriented Analysis

- ✧ The main task is identifying the objects.
- ✧ Also: Relationships between objects.
- ✧ Three strategies:
  - Conceptual Model (concepts as objects)
  - CRC cards (index cards as objects)
  - Analysis Model (Stereotypes as objects)



Next step: Object-Oriented Analysis and Design