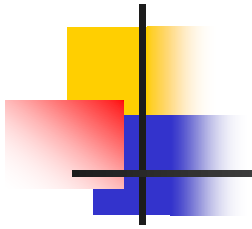
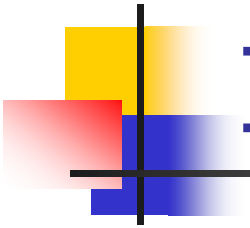
Object Oriented System Design Responsibilities



- Marc Conrad
  - D104 (Park Square Building)
  - Email: Marc.Conrad@beds.ac.uk
  - WWW: http://perisic.com/marc
- This week new:
  - Responsibility patterns
- Or: How to decide who is doing what.



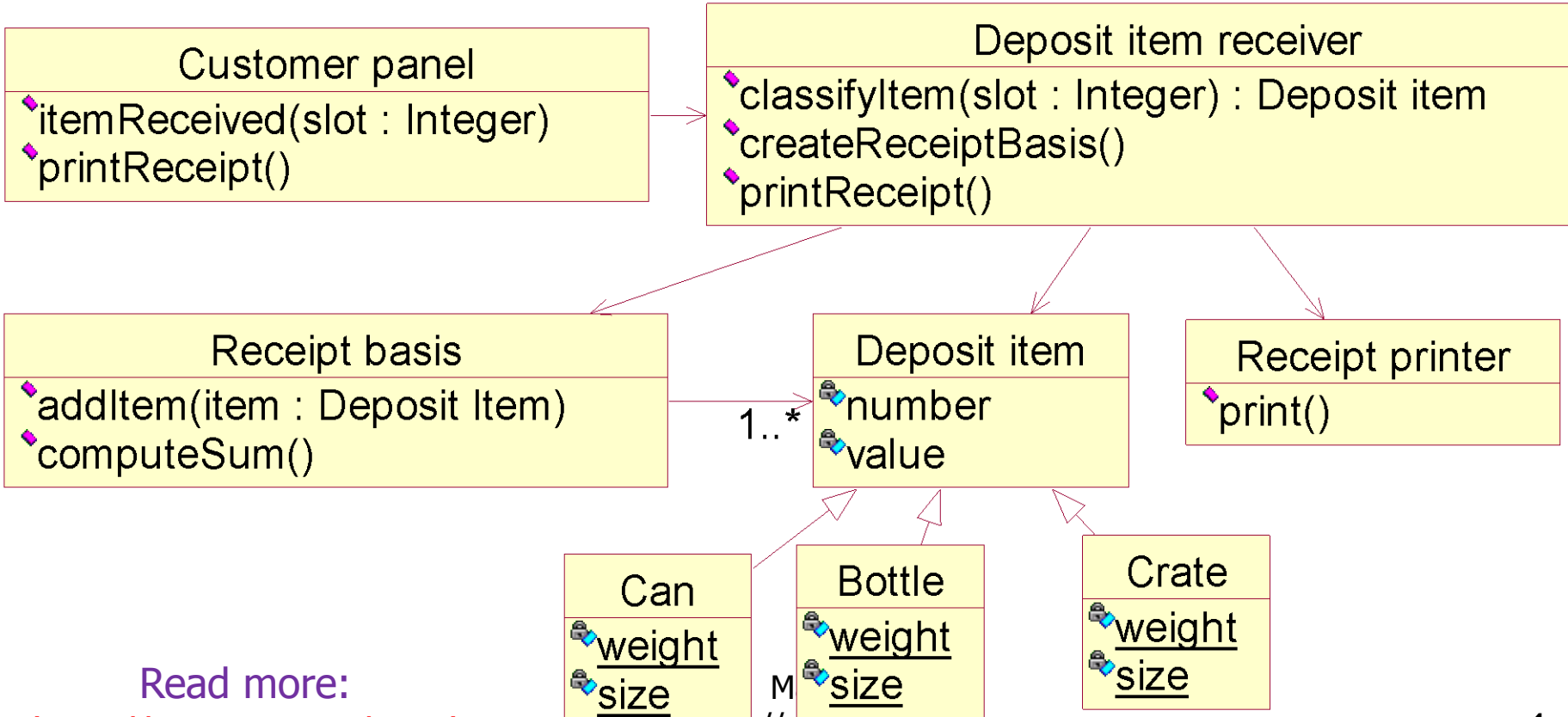
Introduction

- An object-oriented system is composed of objects sending messages to other objects.
- The quality of the overall design depends on which object is doing what.
- That is, the quality depends on how we assign *responsibilities* to the objects.
- Problem: Define "good quality".

There are two types of Responsibilities.

- Knowing
  - about private encapsulated data
  - about related objects
  - about things it can derive or calculate
- Doing
  - doing something itself
  - initiating action in other objects
  - controlling and coordinating activities in other objects

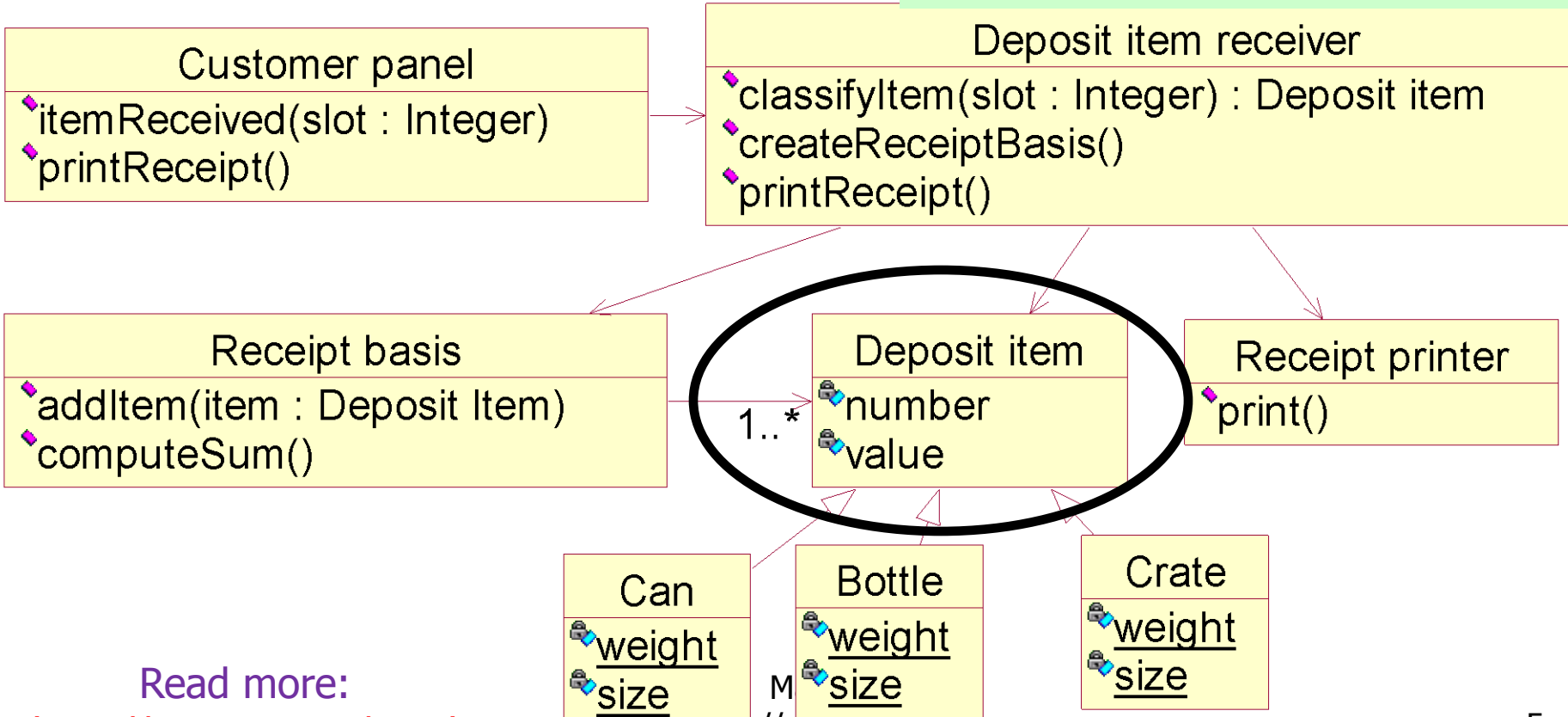
Example – the Recycling machine – Knowing and Doing



Read more:
http://perisic.com/oosd

- *Deposit item* knows about private data as number and value

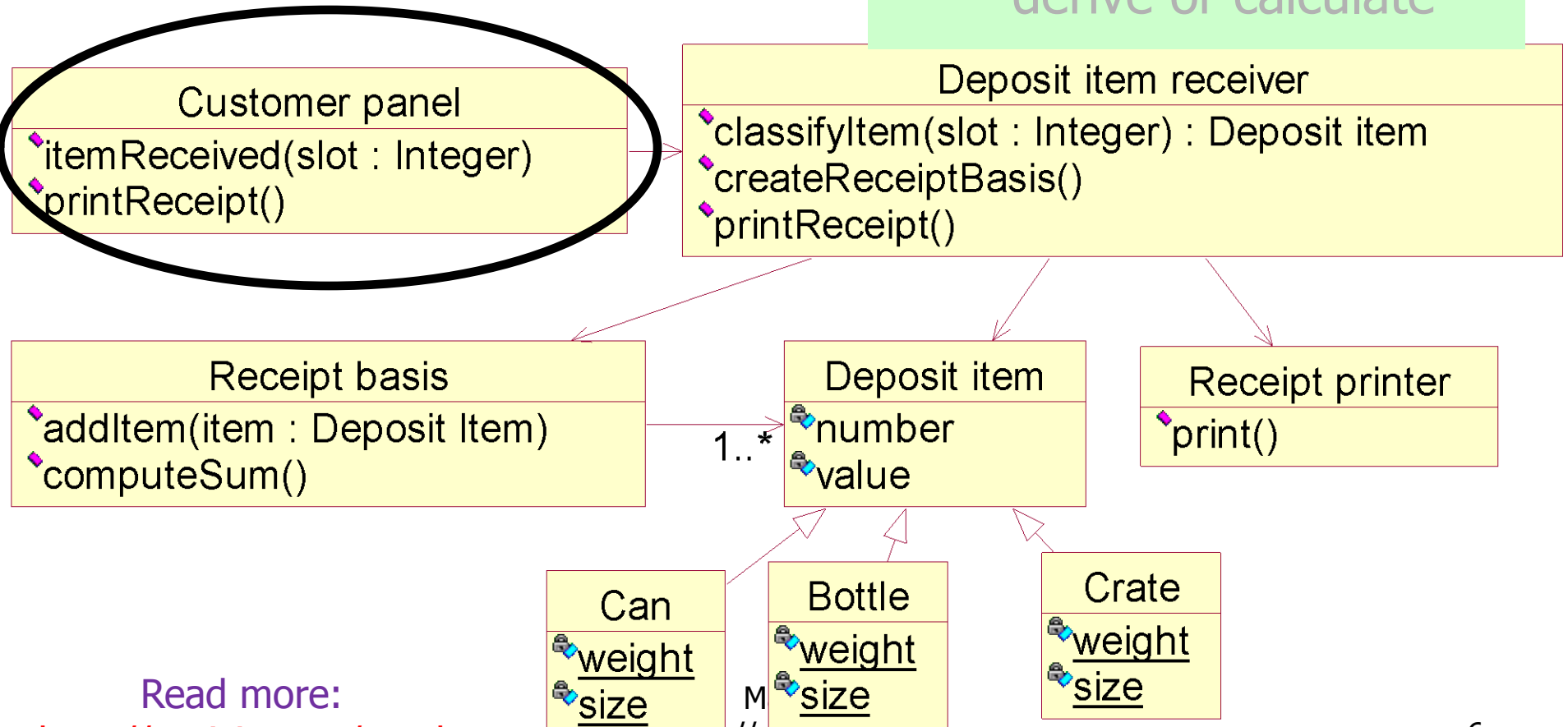
- Knowing
  - about private encapsulated data
  - about related objects
  - about things it can derive or calculate



Read more:
http://perisic.com/oosd

- *Customer panel* knows about the Deposit item receiver where it sends it messages to.

- Knowing
  - about private encapsulated data
  - about related objects
  - about things it can derive or calculate

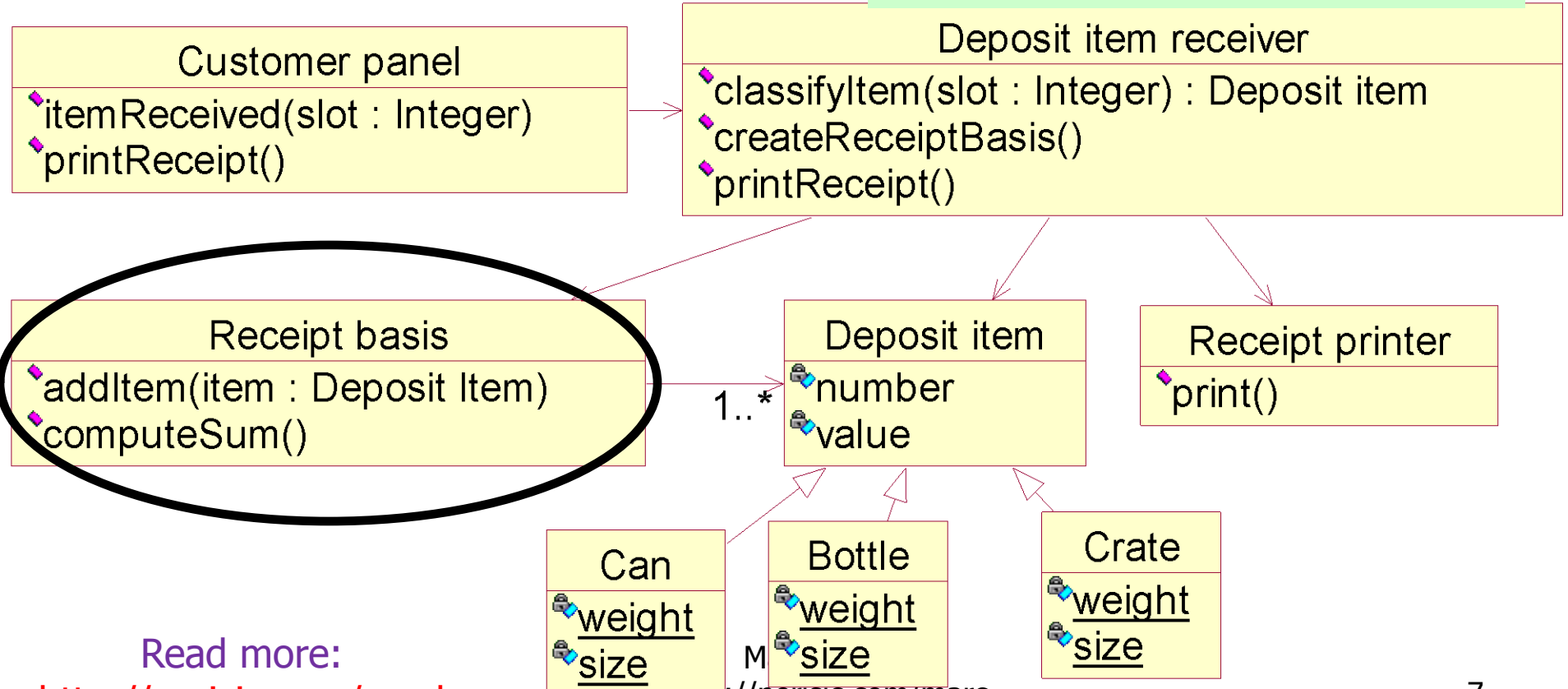


Read more:
http://perisic.com/oosd

- *Receipt basis* knows all the items which have been inserted into the recycling machine and is therefore able to compute the sum of their values.

he I

- Knowing
  - about private encapsulated data
  - about related objects
  - about things it can derive or calculate

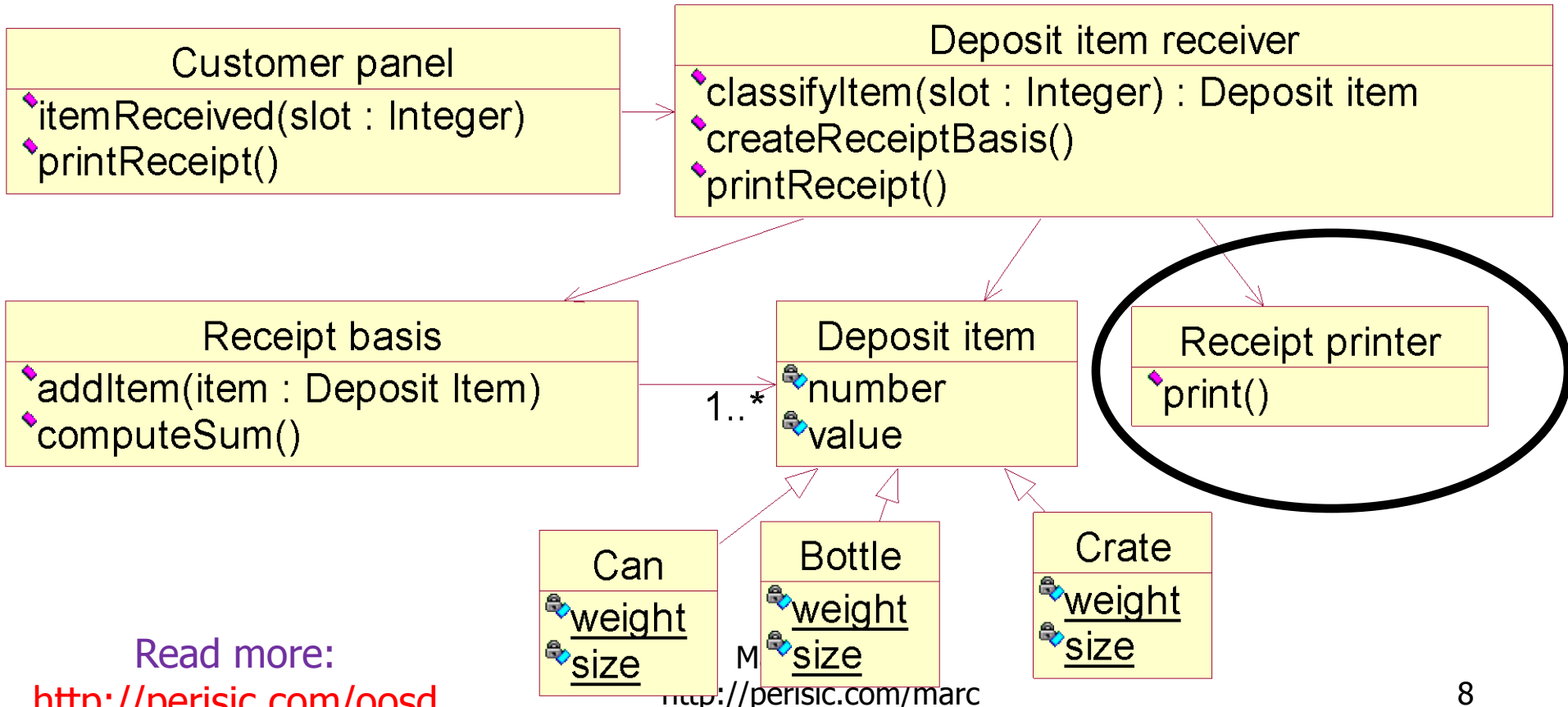


Read more:
http://perisic.com/oosd

Example – th machine

- The Receipt printer does print receipts.

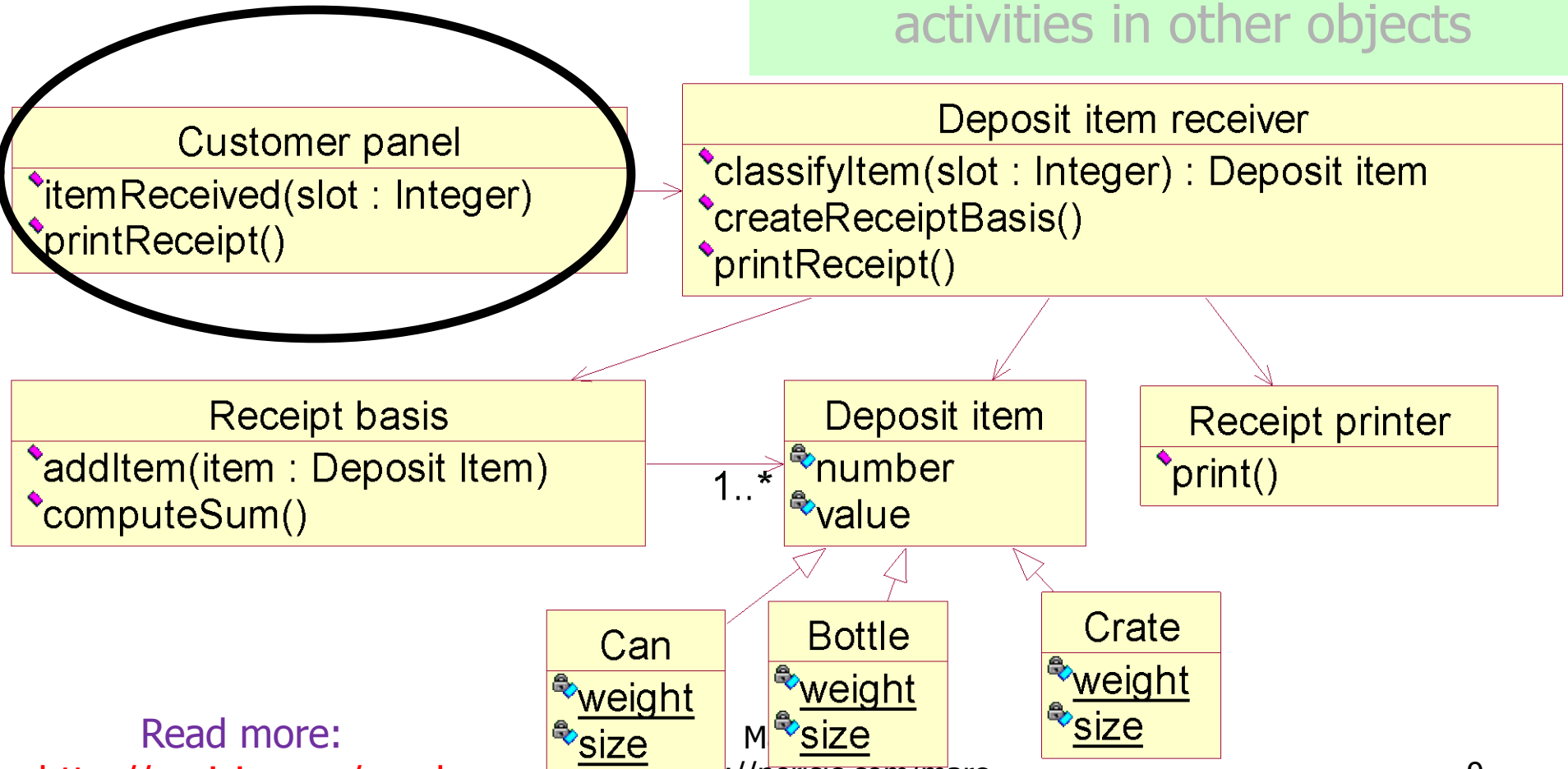
- Doing
  - doing something itself
  - initiating action in other objects
  - controlling and coordinating activities in other objects



Read more:
http://perisic.com/oosd

- The Customer panel initiates the classification and receipt printing action in the Deposit item receiver.

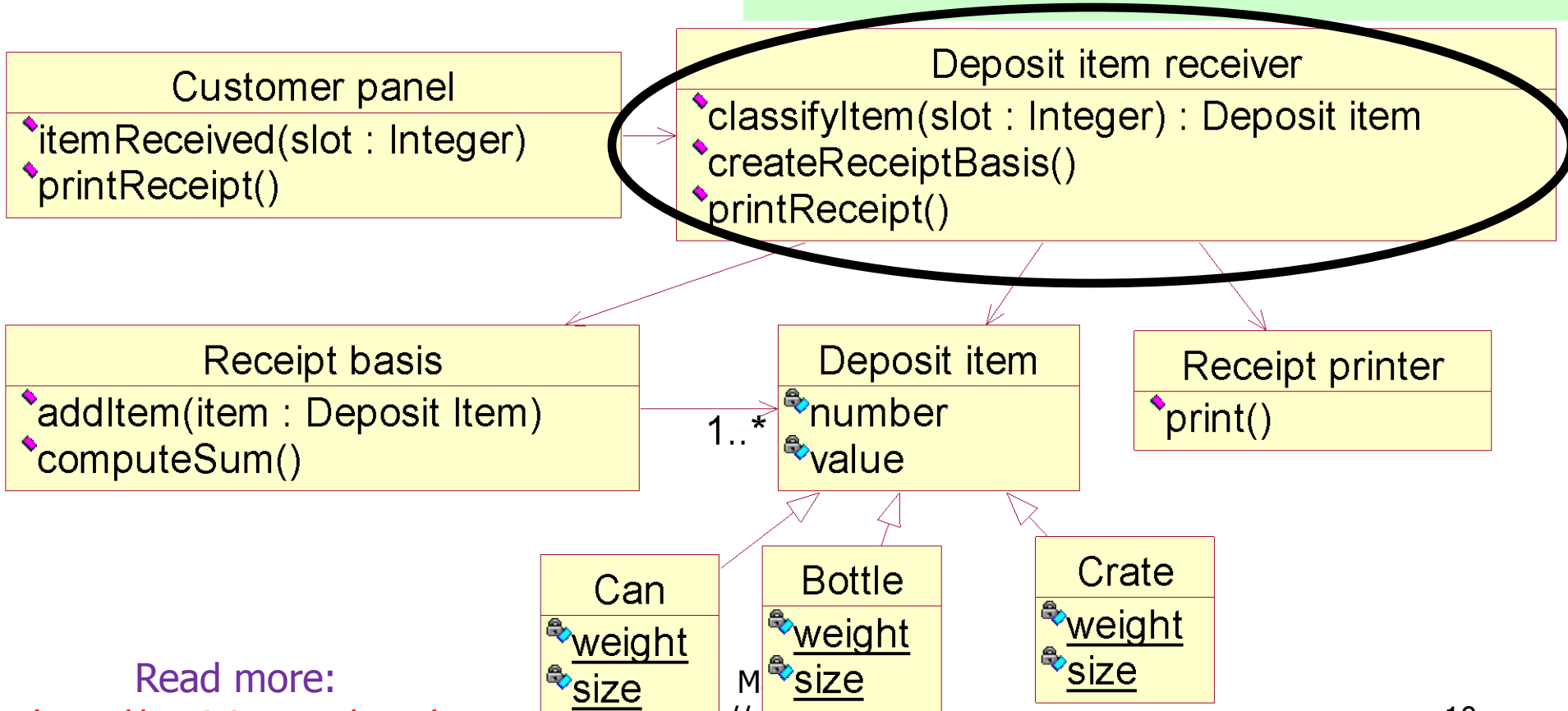
- Doing
  - doing something itself
  - initiating action in other objects
  - controlling and coordinating activities in other objects



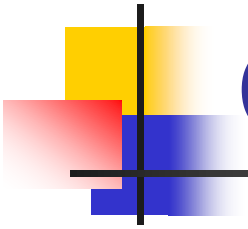
Read more:
http://perisic.com/oosd

- The Deposit item receiver controls the overall system behavior by assigning tasks to other objects (Receipt basis, Receipt printer).

- Doing
  - doing something itself
  - initiating action in other objects
  - controlling and coordinating activities in other objects



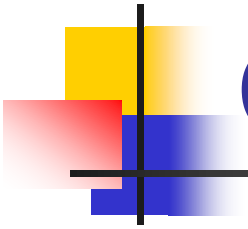
Read more:
http://perisic.com/oosd



Good design – bad design

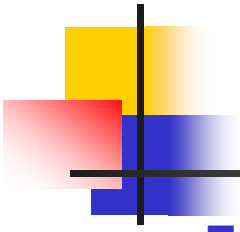
- Consider the following alternative design of the recycling machine.
  - A class responsible for printing and holding the data of bottle and crate.
  - The can class is also responsible for customer input and computing the sum.
  - One more class doing all the rest of the tasks.
- Is *this* a good design???

Read more:
http://perisic.com/oosd



Good design – bad design

- Our feeling says that the previous example is not a good design.
- Is it possible to give this "feeling" a more solid, more objective, more traceable, and more comprehensible foundation?
- Answer: Yes, by using patterns.



GRASP - patterns

- GRASP stands for General Responsibility Assignment Software Patterns.
- GRASP can be used when designing interaction (sequence) diagrams and class diagrams.
- GRASP try to formalize "common sense" in object oriented design.
- They do not usually contain "new" ideas. They try to codify existing knowledge and principles.

Read more:
http://perisic.com/oosd

The GRASP patterns are

- Expert
- Creator
- High Cohesion
- Low Coupling

- (will be discussed in detail in this lecture)

- Controller
- Polymorphism
- Pure Fabrication
- Indirection
- Don't talk to Strangers

- (will not be discussed in detail in this lecture)

Read more:
http://perisic.com/oosd

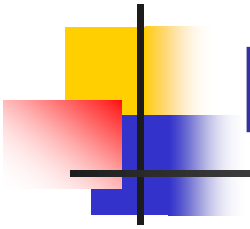
GRASP – patterns for responsibilities

Expert

- ✓ Assign a responsibility to the information expert – the class that has the *information* necessary to fulfil the responsibility

Read more:
http://perisic.com/oosd

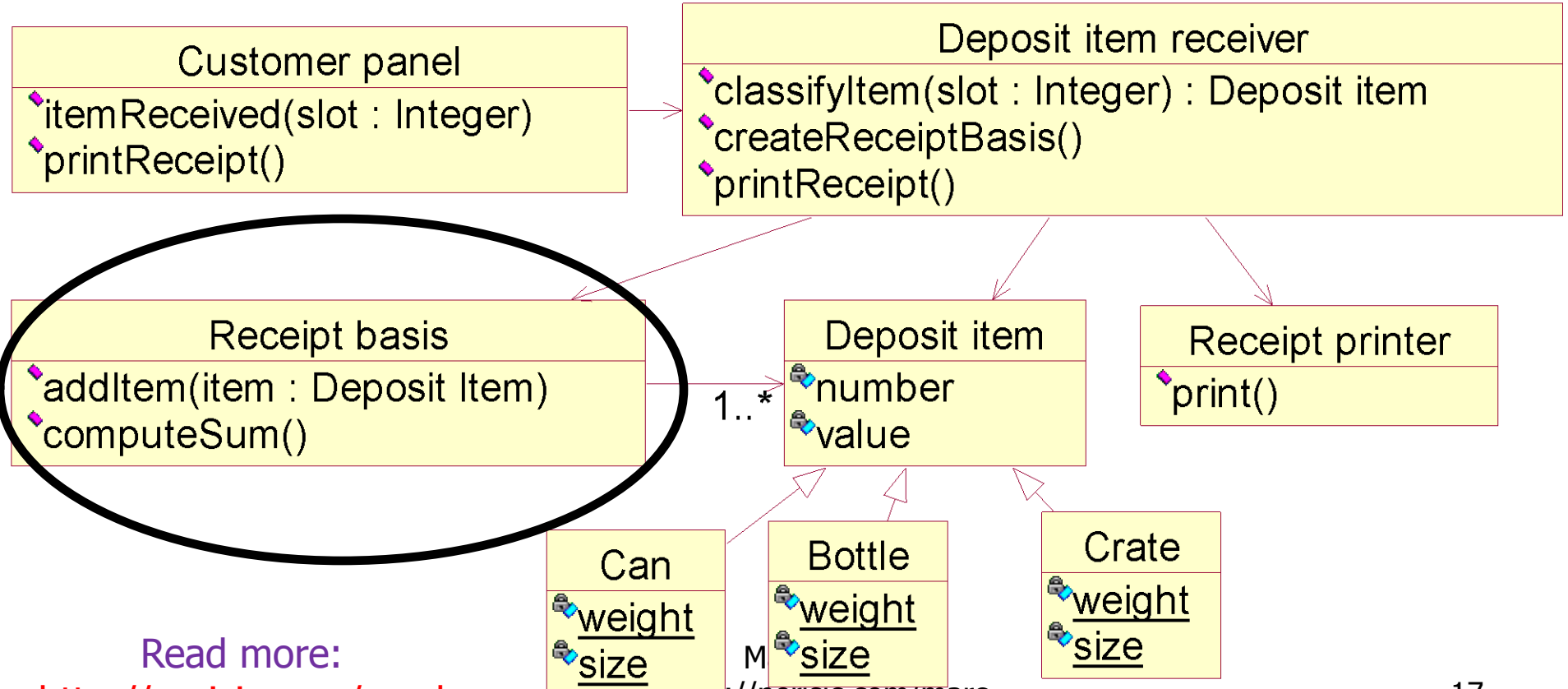
Discussion of the Expert Pattern



- Expert is *the* basic guiding principle in object-oriented design.
- Expert leads to designs where a software object does those operations which are normally done *to* the real-world thing it represents ("Do it Myself")
- Real-world example:
  - When going for medical treatment - which person would you ask for an appointment? The cleaner, the receptionist, or the doctor?

Read more:
http://perisic.com/oosd

- The Receipt basis aggregates all Deposit item objects which have been inserted in the machine. So it is an *Expert* for computing the total of the values of these items.



Read more:
http://perisic.com/oosd

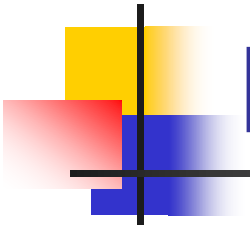
GRASP – patterns for responsibilities

Creator

- Assign class B the responsibility to create an instance of class A if one of the following is true:
  - B *aggregates* A.
  - B *contains* A.
  - B *records* instances of A objects.
  - B *closely uses* A objects.
  - B *has the initialising data* that will be passed to A when it is created.

Read more:
http://perisic.com/oosd

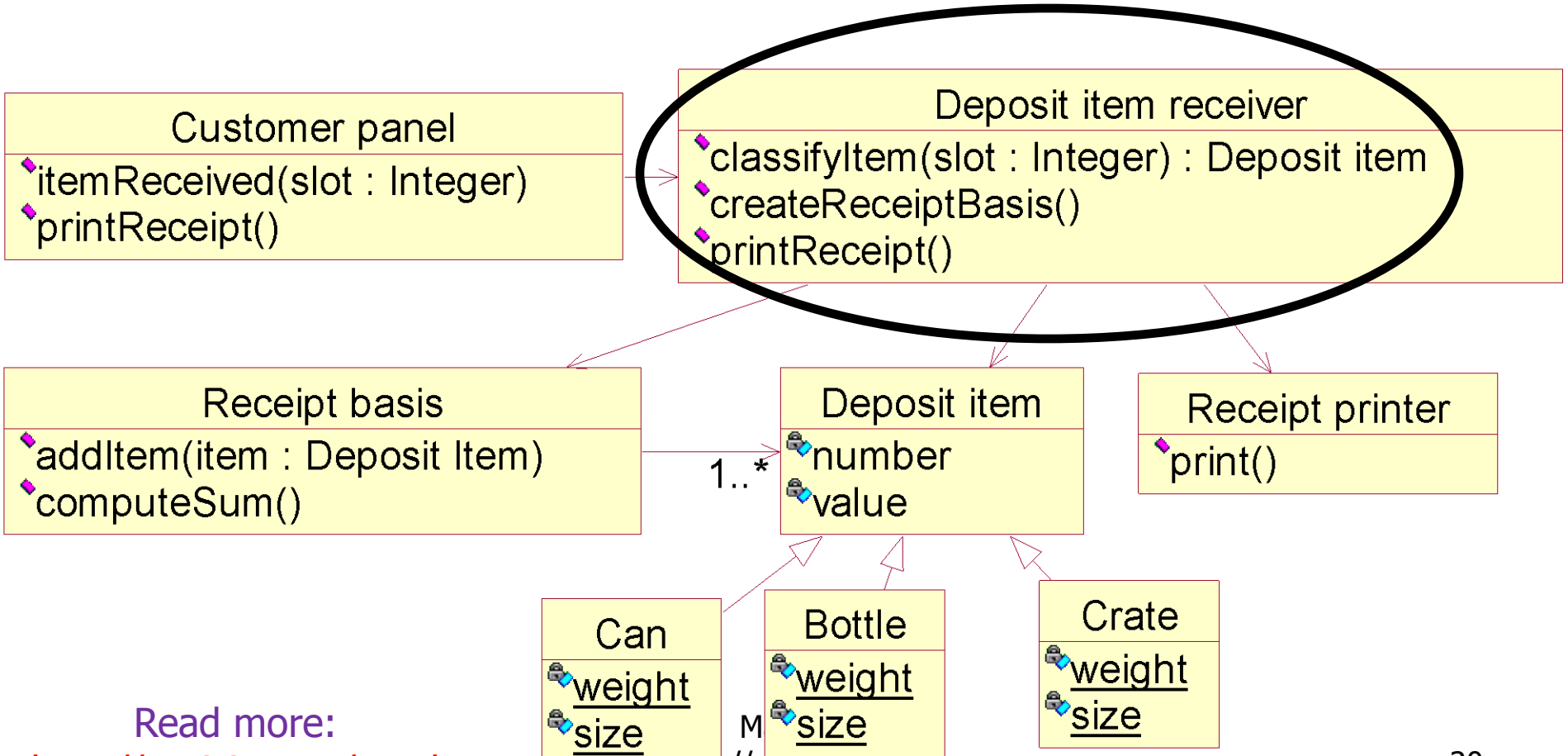
Discussion of the Creator Pattern.



- The creation of objects is one of the most common activities in an object-oriented system.
- This pattern is useful to find out who should be responsible for creating objects.
- The last point (B has initialising data of A) is actually an example of the Expert pattern (B is an expert with respect to creating A).
- In an Aggregation the lifetime of the part is usually the same as the lifetime of the whole. So the idea that the whole creates the part is straightforward.

E C

- The Deposit item receiver has all the necessary data for creating a Deposit item object.



Read more:
http://perisic.com/oosd

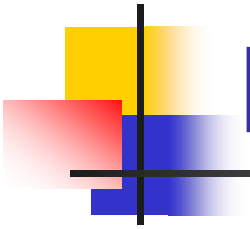
GRASP – patterns for responsibilities

Low Coupling

- Assign a responsibility so that coupling remains low.
  - *Coupling* is a measure of how strongly one class is
    - connected to,
    - has knowledge of, or
    - relies upon

  other classes.

Read more:
http://perisic.com/oosd

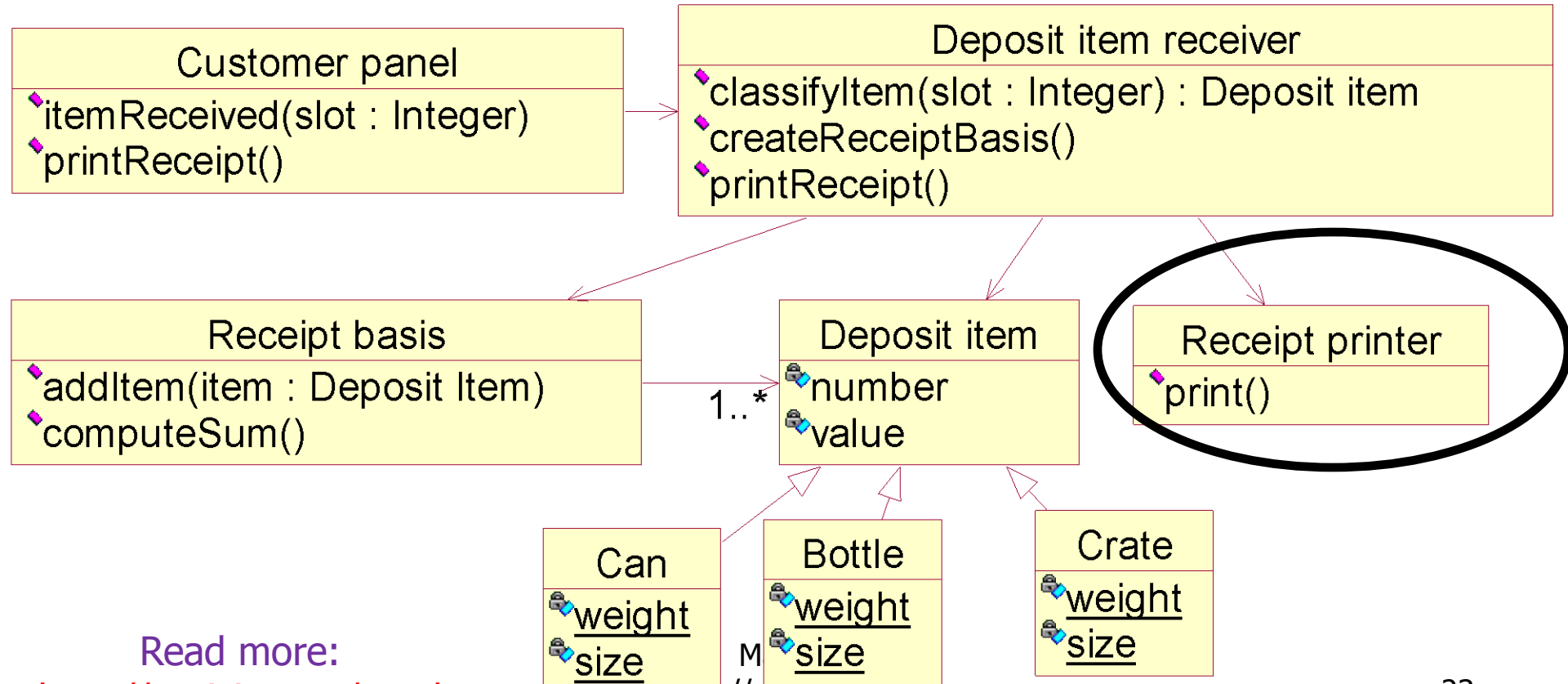


Discussion of Low Coupling

- Low Coupling is an *evaluative pattern* which a designer applies while evaluating all design decisions.
- Coupling happens in the same forms as visibility: local, global, as a parameter, as an attribute.
- A subclass is strongly coupled to its superclass, so subclassing needs to be considered with care!
- Low Coupling supports reuseability, so classes which are inherently very generic in nature should have especially low coupling.

E
Id

- The Receipt printer is not dependent on other objects in this design.
- Similarly the Deposit item, but it is structurally dependent on the overall system.

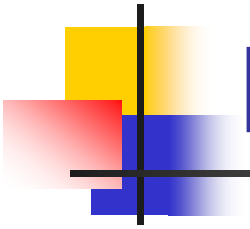


Read more:
http://perisic.com/oosd

GRASP – patterns for responsibilities

High Cohesion

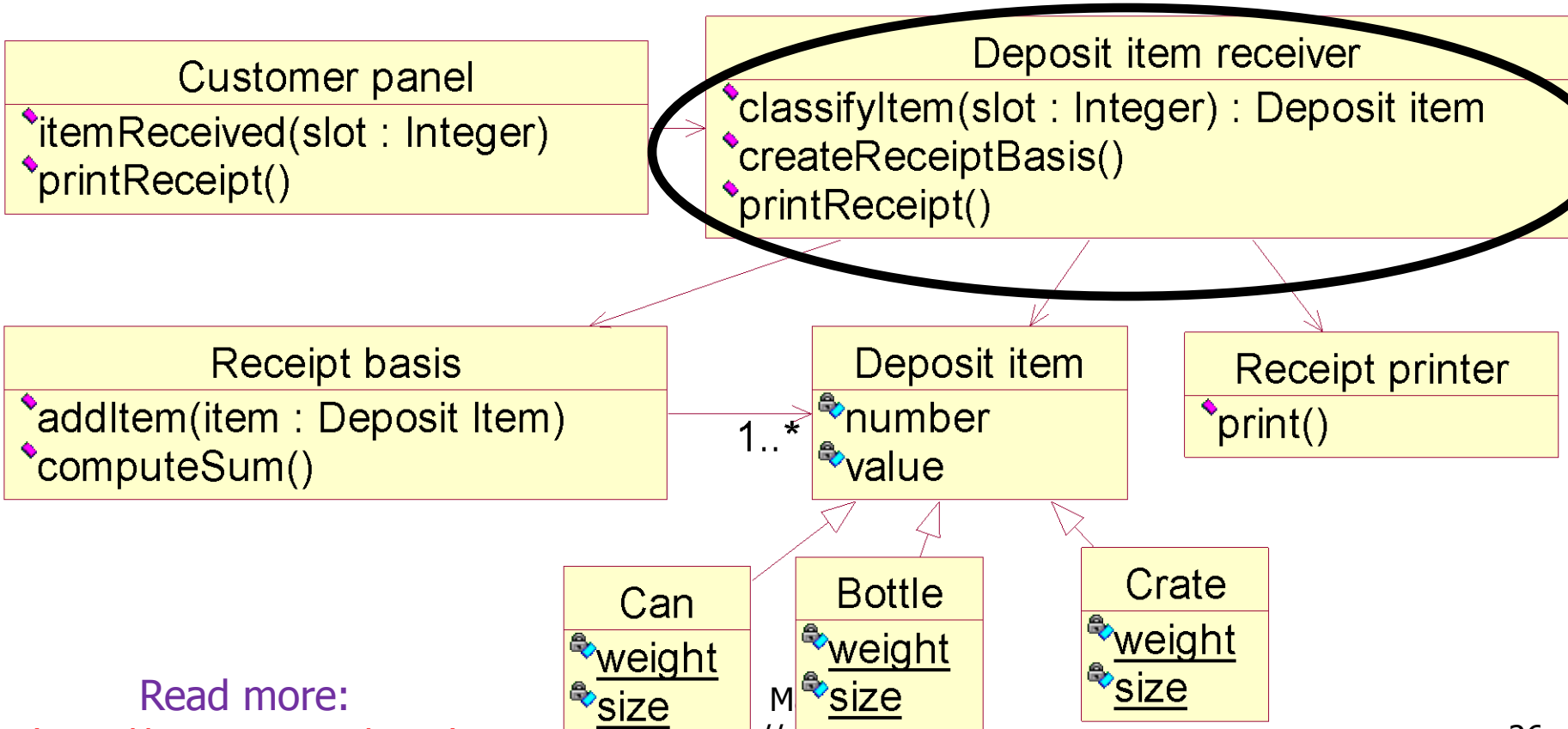
- Assign a responsibility so that cohesion remains high.
  - Here, *cohesion* is a measure of how strongly related and focused the responsibilities of a class are.
  - A class with highly related responsibilities, and which does not do a tremendous amount of work, has high cohesion.



Discussion of High Cohesion

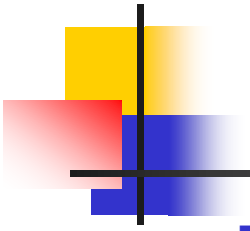
- Benefits:
  - Clarity and ease of comprehension of the design is increased.
  - Maintenance and enhancements are simplified.
  - Low coupling is often supported.
- Rule of thumb:
  - A class with high cohesion has a relatively small number of methods, with highly related functionality, and does not too much work.

- The Deposit item receiver has two unrelated tasks, namely classifying the items and printing the receipt.
- Solutions:
  - Split the class in two, *or*
  - Assign the "printReceipt" responsibility to someone else (e.g. the Receipt basis).



Read more:
http://perisic.com/oosd

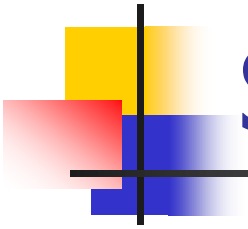
GRASP – patterns for responsibilities Controller, Polymorphism, Pure Fabrication, Indirection, Don't Talk to Strangers



- Controller
  - Who should handle a system event?
- Polymorphism
  - How to handle alternatives based on type?
- Pure Fabrication
  - Who, when you are desperate?
- Indirection
  - How to de-couple objects?
- Don't Talk to Strangers
  - To whom should messages be sent?

(see *Larman*, Applying UML and patterns for details)

Read more:
http://perisic.com/oosd



Summary - patterns

- Object Oriented Design is about responsibilities.
- Patterns help us to identify and assign responsibilities to objects.
- The main (GRASP) patterns are *Expert, Creator, Low Coupling, High Cohesion.*
- Note that we also may identify responsibilities in the analysis phases (e.g. CRC cards, stereotypes, …)

Read more:
http://perisic.com/oosd