



Advanced Software Design Design Patterns

Gary Stewart
gstewart@cs.uct.ac.za
slides taken from prof. edwin blake

Class Design and Unified Process Terminology

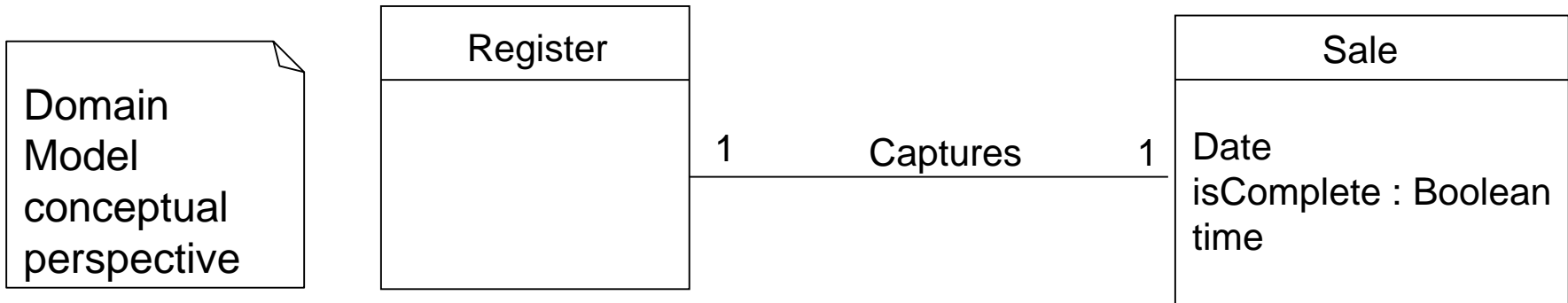
2

- Typical information in a Design Class Diagram includes:
 - ▣ Classes, associations and attributes
 - ▣ Interfaces (with operations and constants)
 - ▣ Methods
 - ▣ Attribute type information
 - ▣ Navigability
 - ▣ Dependencies
- The Class Design depends upon the Domain Model and interaction diagrams.
- The UP defines a Design Model which includes interaction and class diagrams.

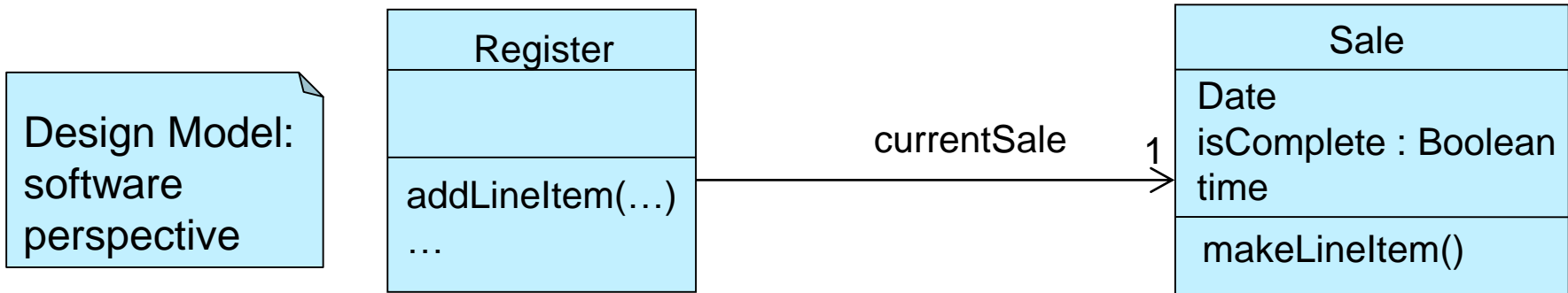
Domain Model versus Design Model

3

- Domain Model is the analysis class diagram
- Don't show methods

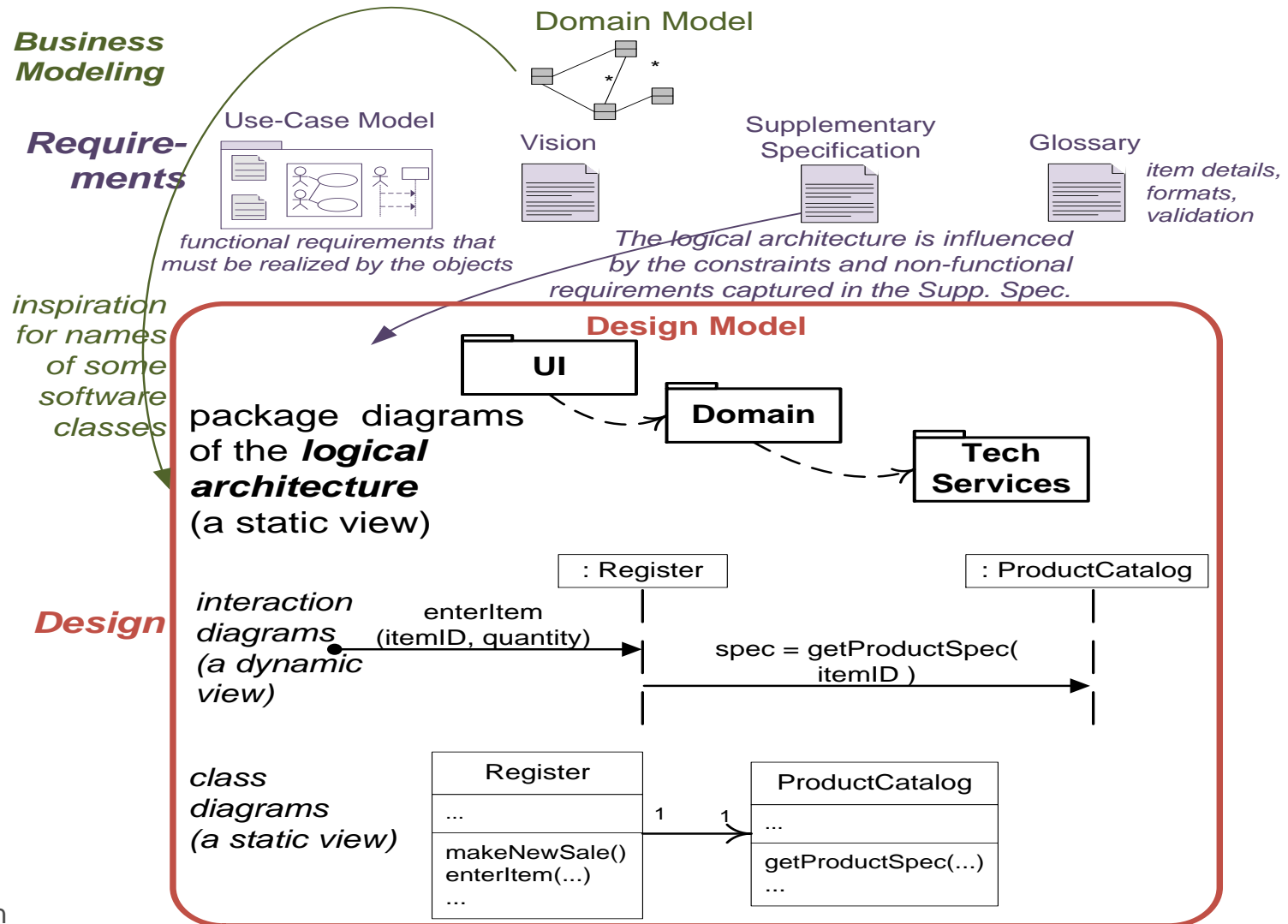


- Design Model shows methods and visibility (arrowhead on association)
- Register has reference to Sale; Sale does not have reference to Register



Sample UP Artefact Relationships

4





5

RESPONSIBILITY-DRIVEN DESIGN

Responsibility-Driven Design

GRASP

Use Case Realizations

GoF Patterns

Conclusion

Responsibility-Driven Design (RDD)

6

- **RDD:** software objects have responsibilities (an abstraction)
 - ▣ “a *Sale* is responsible for creating *SalesLineItems*” (doing)
 - ▣ “a *Sale* is responsible for knowing its total” (knowing).
- Metaphor for thinking about OO design
- Big responsibilities take lots of classes and methods
 - ▣ “provide access to relational databases”: subsystem with 100 classes & 1000 methods
- Small responsibilities may take one method.
 - ▣ “create a *Sale*”: one method in one class
- A responsibility is not a method, but methods fulfil responsibilities.

Responsibilities and Methods

7

Object design is about identifying classes and objects, and their methods, and how they interact.

- Responsibilities relate to the obligations of an object.
- Two types of responsibilities:
 - ▣ Doing:
 - Doing something itself (e.g. creating an object, doing a calculation)
 - Initiating action in other objects.
 - Controlling and coordinating activities in other objects.
 - ▣ Knowing:
 - Knowing about private encapsulated data.
 - Knowing about related objects.
 - Knowing about things it can derive or calculate.

RDD and Collaboration

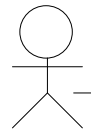
8

- Responsibilities are implemented with methods that either act alone or collaborate with other methods and objects.
 - ▣ The *Sale* class might define one or more methods to know its total; say, a method named *getTotal*.
 - ▣ The *Sale* may collaborate with other objects, such as sending a *getSubtotal* message to each *SalesLineItem* object asking for its subtotal.

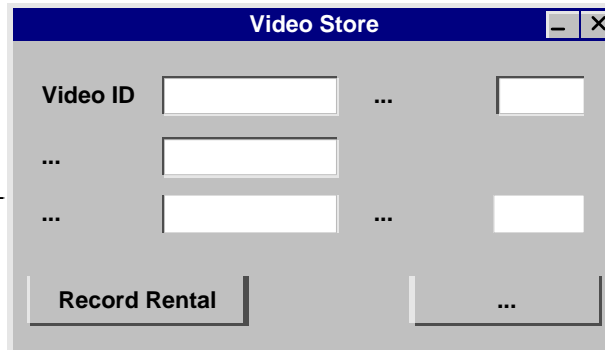
Motivation: Introduction

9

Presentation



Clerk



Application Logic

Now what happens?

???

rent(videoID) ↓

- Now what happens?
- What object should receive this message?
- What objects should interact to fulfil the request, and how?

Definition: Responsibilities

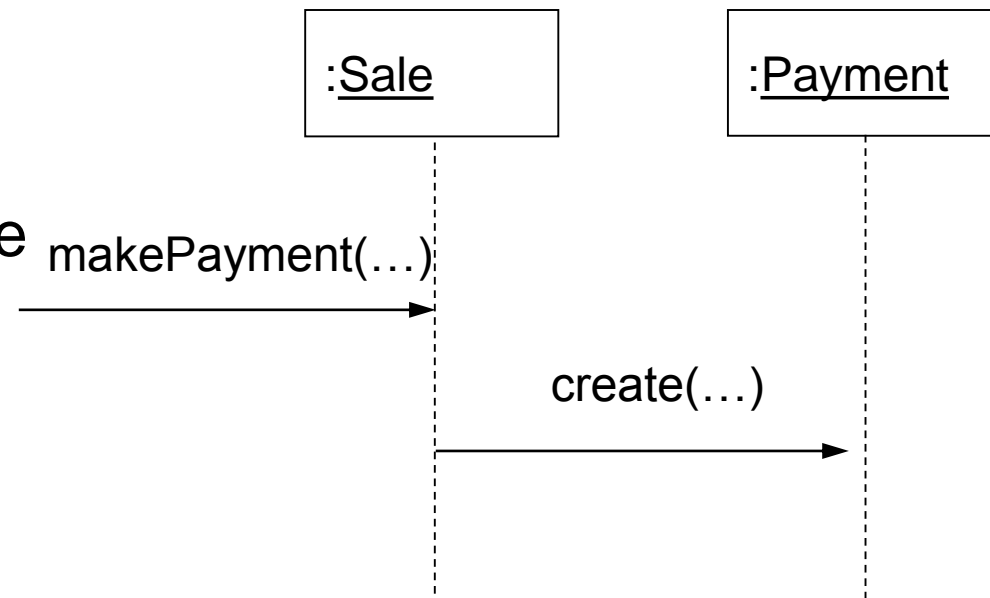
10

- Responsibilities are an abstraction.
 - ▣ The responsibility for persistence.
 - Large-grained responsibility.
 - ▣ The responsibility for the sales tax calculation.
 - More fine-grained responsibility.
- Responsibilities are implemented with methods in objects.
 - ▣ 1 method in 1 object
 - ▣ 5 methods in 1 object
 - ▣ 50 methods across 10 objects

Responsibilities and System Sequence Diagrams

11

- Within the analysis artefacts, a common context where these responsibilities (implemented as methods) are considered is during the creation of sequence diagrams.
- Sale objects have been given the responsibility to create Payments, handled with the makePayment method.



- Designing Objects with Responsibilities
 - ▣ A critical skill is designing or thinking in objects.
 - ▣ This *can* be practiced based on explainable principles.
- Question: What guiding principles to help us assign responsibilities?
- (One) Answer: **G**eneral **R**esponsibility **A**ssignment **S**oftware **P**atterns (GRASP)
 - ▣ Very fundamental, basic principles of object design.
 - ▣ Patterns are *named* problem-solution pairs to common problems, typically showing a popular, robust solution.
 - “Façade” “Information Expert” ...
 - ▣ They provide a *vocabulary* of design.



13

GRASP

Responsibility-Driven Design

General Responsibility Assignment Software Patterns (GRASP)

Information Expert

Creator

Controller

Low Coupling

High Cohesion

Polymorphism

Pure Fabrication

Indirection

Protected Variations (Don't talk to strangers)

Use Case Realizations

GoF Patterns

Conclusion

Patterns

14

- Principles (expressed in *patterns*) guide choices in where to assign responsibilities.
- A pattern is a named ***description of a problem*** and a ***solution*** that can be applied to new contexts;
 - ▣ it provides advice on how to apply it in varying circumstances.
- For example,
 - ▣ **Pattern_name:** Information Expert
 - ▣ **Problem:** What is the most basic principle by which to assign responsibilities to objects?
 - ▣ **Solution:** Assign a responsibility to the class that has the information needed to fulfil it.

Four elements of Pattern Templates

15

□ Name

- ▣ increases our design vocabulary.
- ▣ Frequently see “Also Known As”,
 - indication of a naming problem

□ Problem

- ▣ describes problem, inherent trade-offs and context

□ Solution

- ▣ General description of how to solve the problem.
 - abstraction of an entire family of similar solutions

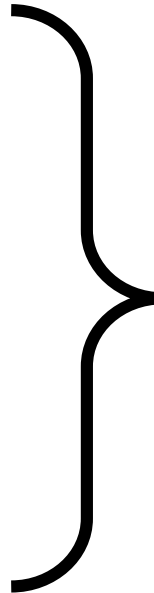
□ Consequences — usually required

- ▣ each solution has trade-offs and consequences.
- ▣ solutions can cause or amplify other problems.
 - Costs and benefits should be compared against

Nine GRASP Patterns

16

- Information Expert (Expert)
- Creator
- Controller
- Low Coupling
- High Cohesion
- Polymorphism
- Pure Fabrication
- Indirection
- Protected Variations
(Don't talk to strangers)



Memorize!

Information Expert (or Expert)

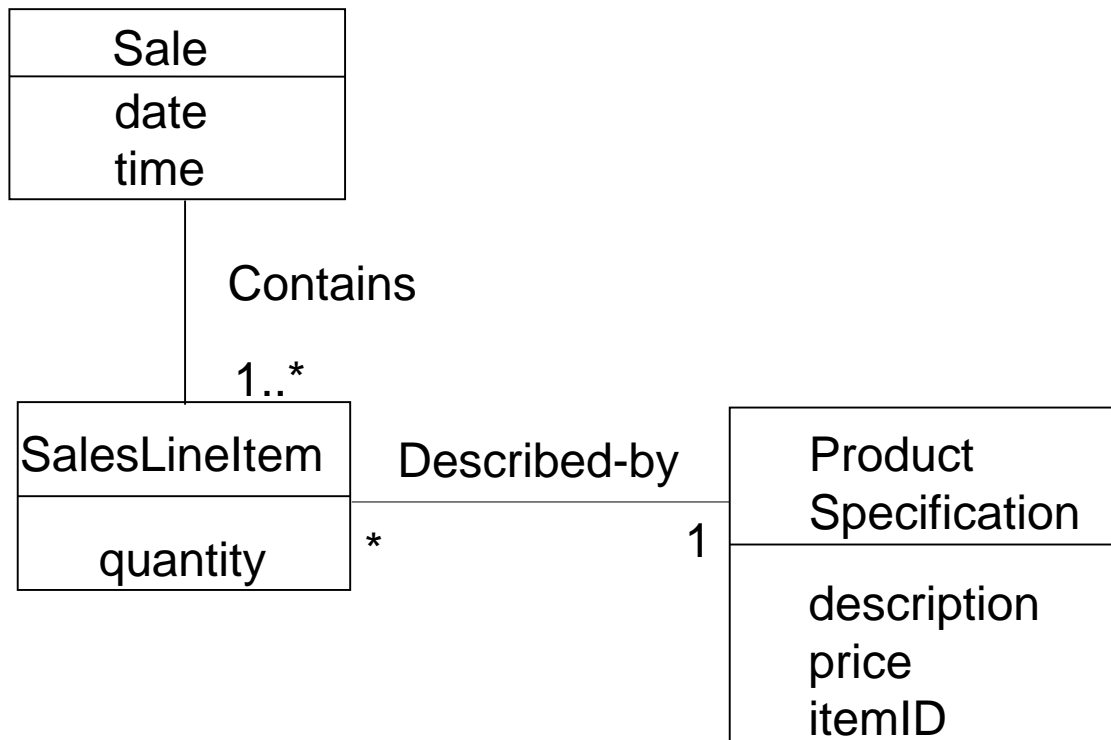
17

- What is most basic, general principle of responsibility assignment?
- ⇒ Assign a responsibility to the information expert — the class that has the information necessary to fulfil the responsibility.
 - ▣ “That which has the information, does the work.”

Information Expert Exercise

18

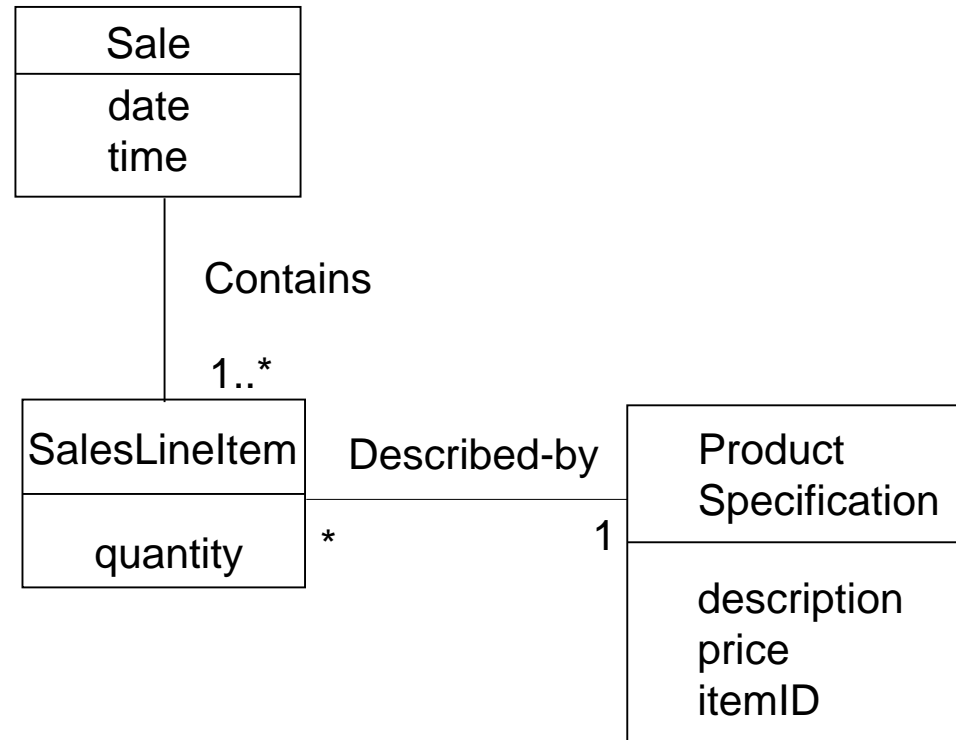
- What software object calculates sales total?
 - What information is needed to do this?
 - What object or objects has the majority of this information.



Information Expert Exercise Answer

19

- It is necessary to know about all the SalesLineItem instances of a sale and the sum of the subtotals.
- A Sale instance contains these, i.e. it is an *information expert* for this responsibility.



Information Expert

20

- To fulfil the responsibility of knowing and answering the sale's total, three responsibilities were assigned to three design classes
- The fulfilment of a responsibility often requires information that is spread across different classes of objects. This implies that there are many “partial experts” who collaborate in the task.

Class	Responsibility
Sale	Knows Sale total
SalesLineItem	Knows line item total
ProductSpecification	Knows product price

Creator

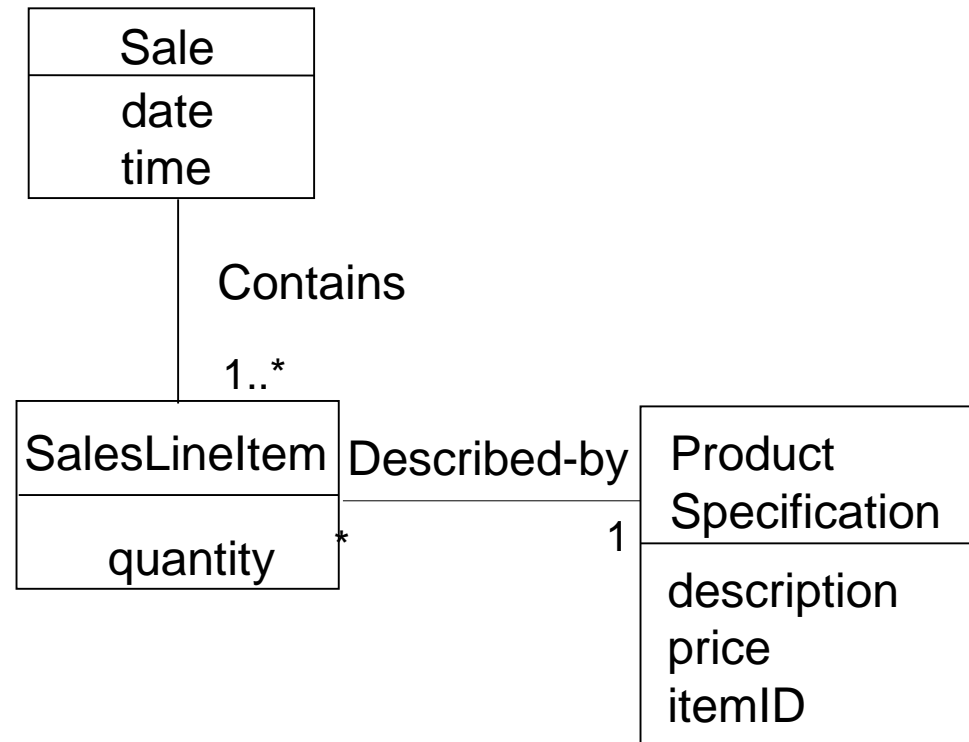
21

- **Problem:** Who should be responsible for creating a new instance of some class?
- **Solution:** Assign class C the responsibility to create an instance of class X if one or more of the following is true:
 - ▣ C aggregates X objects.
 - ▣ C contains X objects.
 - ▣ C records instances of X objects.
 - ▣ C closely uses X.
 - ▣ C has the initializing data that will be passed to X when it is created (thus C is an Expert with respect to creating X).
- The more the better.

Creator

22

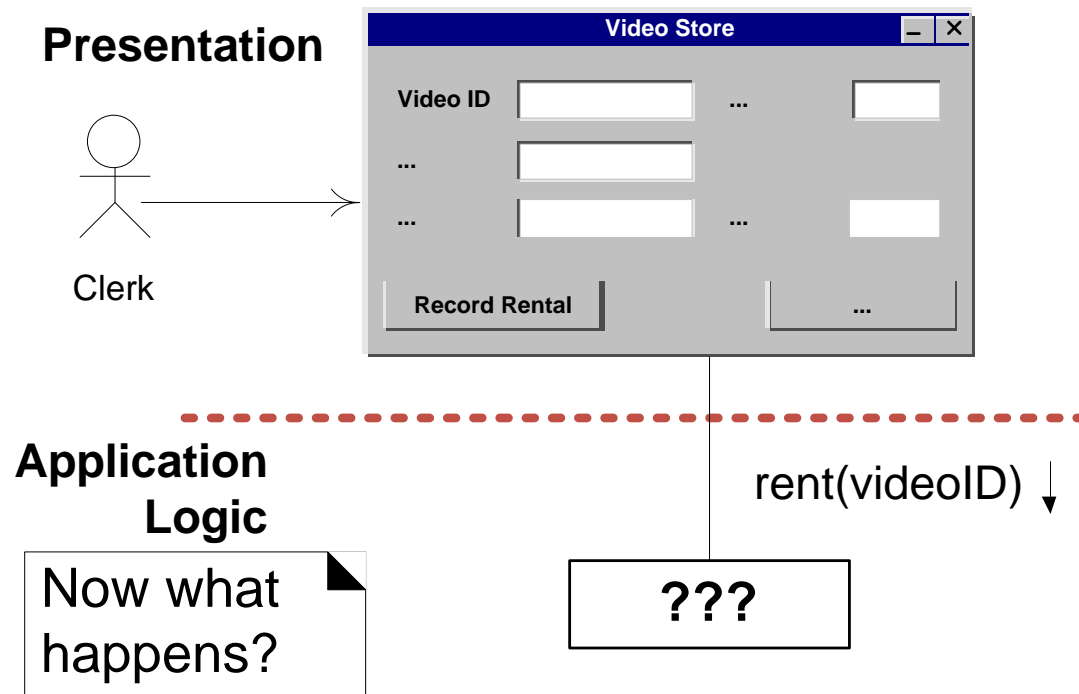
- In the POS application, who should be responsible for creating a SalesLineItem instance?
- Since a Sale contains many SalesLineItem objects, the Creator pattern suggests that Sale is a good candidate.



Controller

23

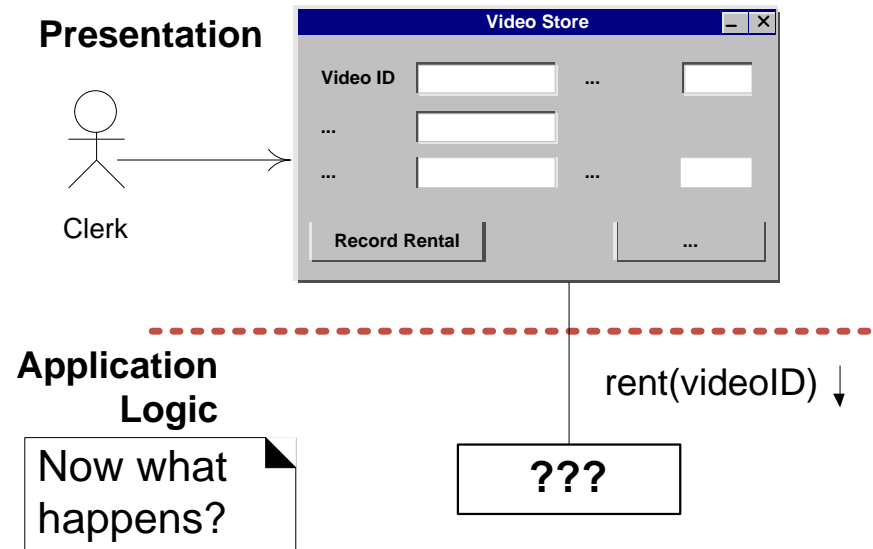
- What object in the domain (or application coordination layer) receives requests for work from the UI layer?



Controller

24

- Solution: Choose a class whose name suggests:
 - ▣ The overall “system,” device, or subsystem
 - A kind of Façade class
 - ▣ Or, represents the use case scenario or session



Controller

25

- **Problem:** Who should be responsible for handling an input system event?
- **Solution:** Assign the responsibility for receiving or handling a system event message to a class representing one of the following choices:
 - ▣ Represents the overall system.
 - ▣ Represents a use case scenario.
 - ▣ A Controller is a non-user interface object that defines the method for the system operation.
(Note that windows, applets, etc. typically receive events and delegate them to a controller.)

Low Coupling

26

- Coupling: it is a measure of how strongly one element is connected to, has knowledge of, or relies upon other elements.
- A class with high coupling depends on many other classes (libraries, tools).
- Problems because of a design with high coupling:
 - ▣ Changes in related classes force local changes.
 - ▣ Harder to understand in isolation; need to understand other classes.
 - ▣ Harder to reuse because it requires additional presence of other classes.
- **Problem**: How to support low dependency, low change impact and increased reuse?
- **Solution**: Assign a responsibility so that coupling remains low.

Low Coupling

27

- Assume we need to create a Payment instance and associate it with the Sale.
- What class should be responsible for this?
- By Creator, Register is a candidate.

:Register

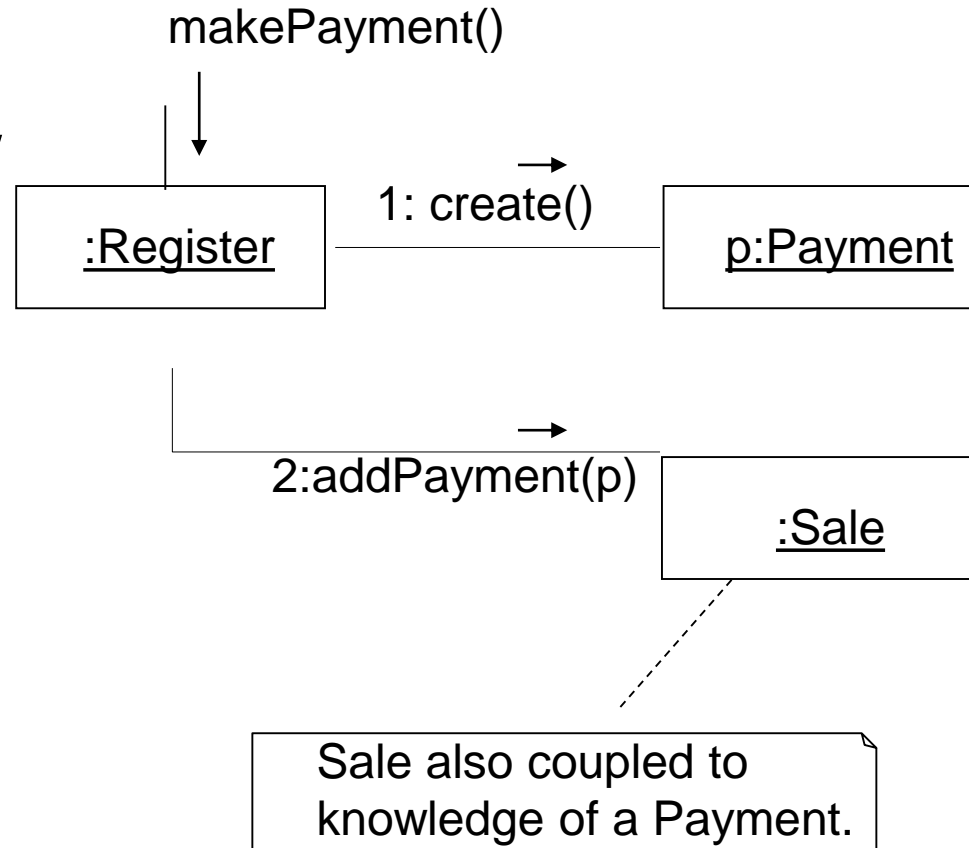
:Payment

:Sale

Low Coupling

28

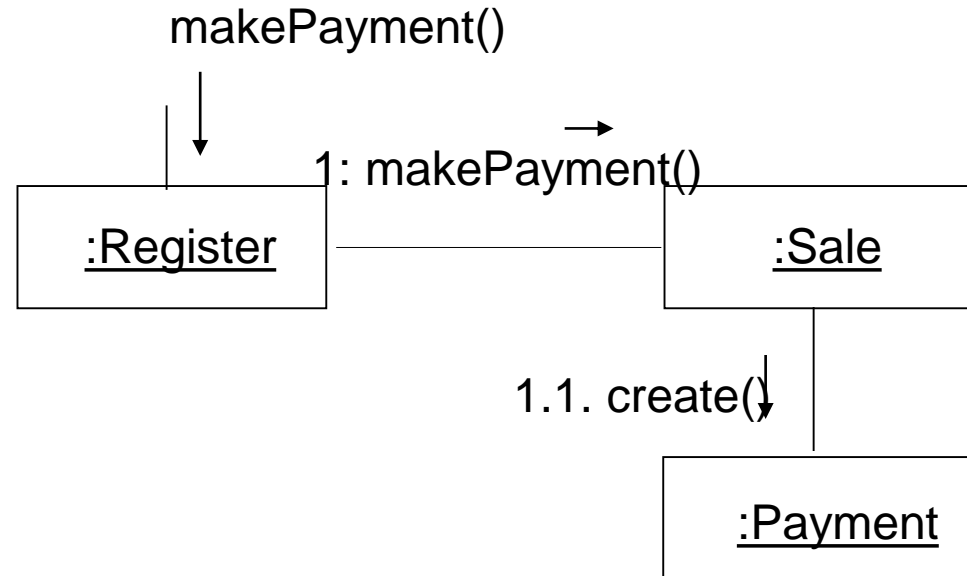
- Register could then send an addPayment message to Sale, passing along the new Payment as a parameter.
- The assignment of responsibilities couples the Register class to knowledge of the Payment class.



Low Coupling

29

- An alternative solution is to create Payment and associate it with the Sale.
- No coupling between Register and Payment.



Low Coupling

30

- Some of the places where coupling occurs:
 - ▣ Attributes: X has an attribute that refers to a Y instance.
 - ▣ Methods: e.g. a parameter or a local variable of type Y is found in a method of X.
 - ▣ Subclasses: X is a subclass of Y.
 - ▣ Types: X implements interface Y.
- There is no specific measurement for coupling, but in general, classes that are generic and simple to reuse have low coupling.
- There will always be some coupling among objects, otherwise, there would be no collaboration.