

GRASP


Curt Clifton

Rose-Hulman Institute of Technology

General,
Responsibility Assignment
Software ~~Patterns~~ Principles

Low Coupling

Coupling



An
evaluative
principle

- A measure of how strongly one element is connected to, has knowledge of, or relies on other elements
- Want low (or weak) coupling
- Several problems with high (strong) coupling...

Example

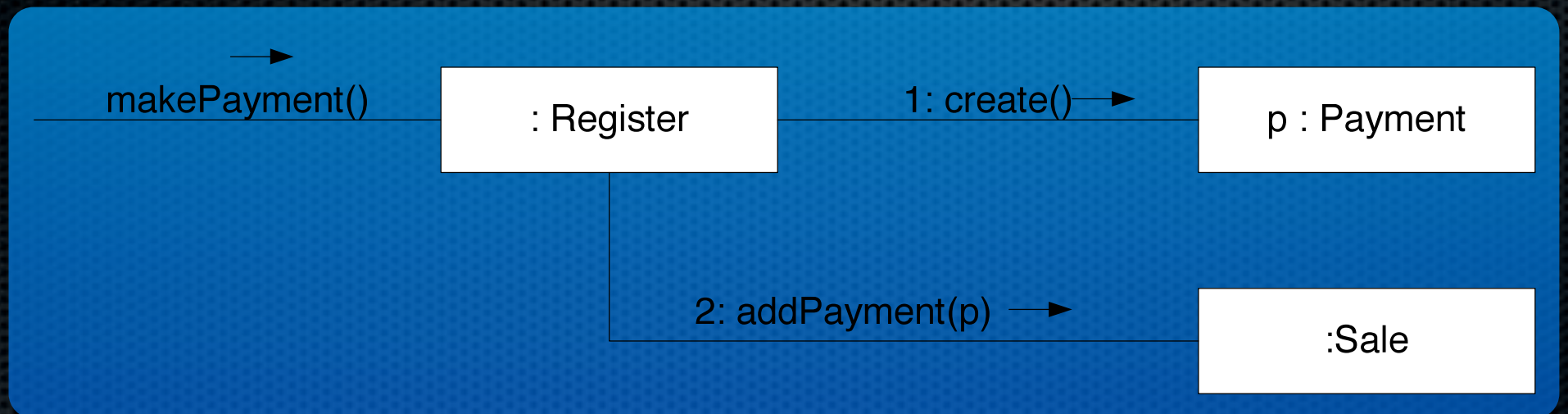
- Suppose we need to create a *Payment* instance and associate it with a *Sale*
- Who should be responsible?

Payment

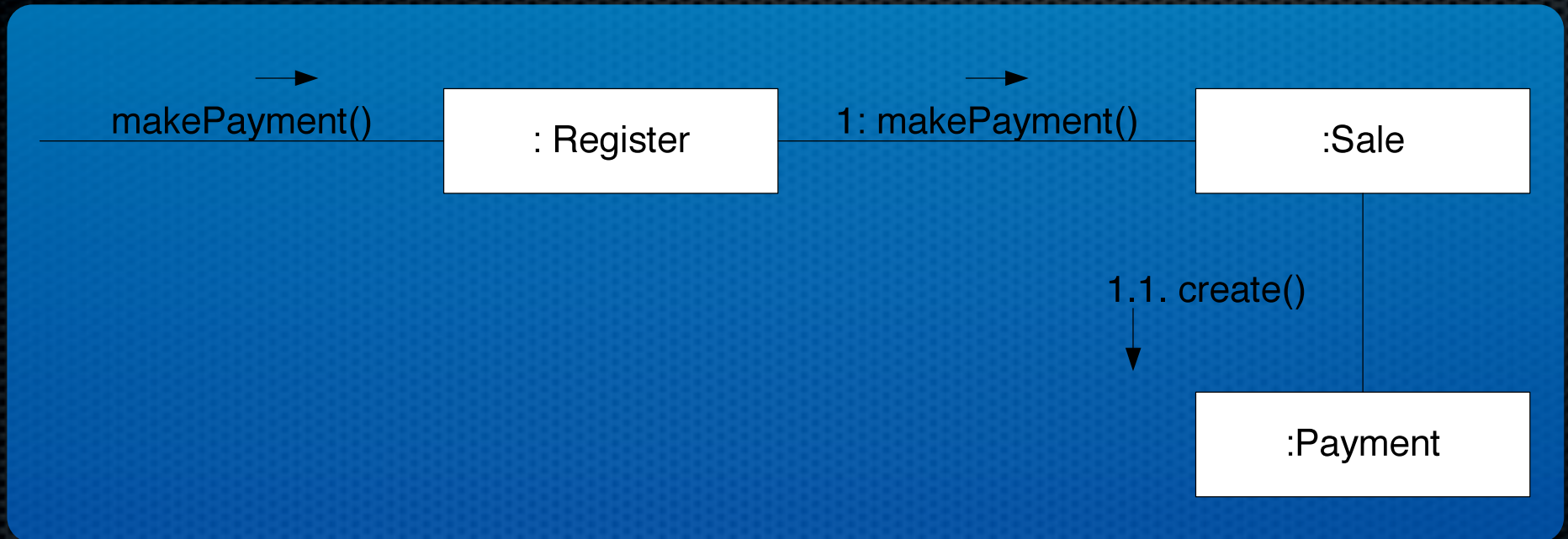
Register

Sale

Option 1



Option 2



Lower Coupling?



Common Couplings

- *A* has an attribute of type *B*
- *A* calls a static method of *B*
- *A* has a method with a parameter or variable of type *B*
- *A* implements an interface *B*
- *A* is a subclass of *B*




Very strong coupling

Pick Your Battles

- ✦ Coupling to stable, pervasive elements isn't a problem
 - ✦ E.g., *java.util.ArrayList*
- ✦ Coupling to unstable elements can be a problem
 - ✦ Unstable interface, implementation, or presence
- ✦ Clearly can't eliminate coupling completely!

High Cohesion

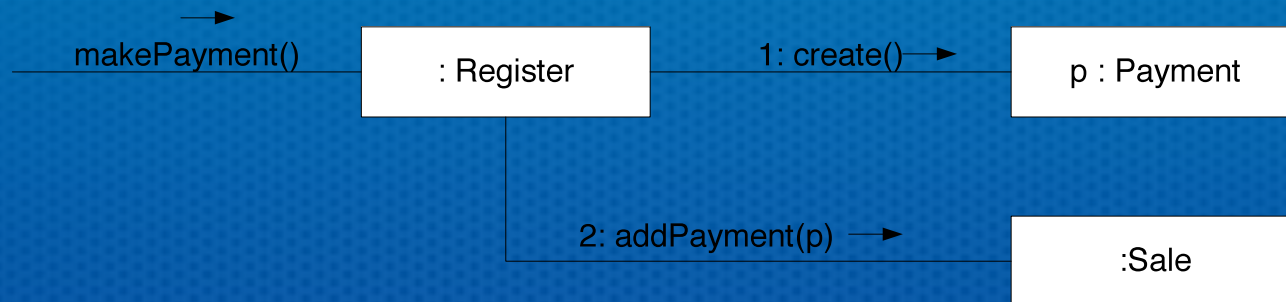
Cohesion



An
evaluative
principle

- A measure of how strongly related and focused the responsibilities of a class (or method or package...) are
- Want high cohesion
- Several problems with low cohesion...

Higher Cohesion?



Guideline

- ✦ A highly cohesive class...
 - ✦ Has a small number of highly related methods
 - ✦ Does not do “too much” work

Contraindications

- ✦ Sometimes lower cohesion is necessary for efficiency
 - ✦ E.g., *setEmployeeData*
vs. *setName*, *setSalary*, and *setHireData*
in a networked application

Information Expert

Information Expert

- ✦ **Problem:** What is a general principle of assigning responsibilities?
- ✦ **Solution:** Assign a responsibility to the class that has the necessary information

perhaps the most
general principle

Where do we look for classes?

- ✦ In the Design model if the relevant classes are there
- ✦ Otherwise:
 - ✦ Look to Domain model for motivation,
 - ✦ then add classes to the Design model

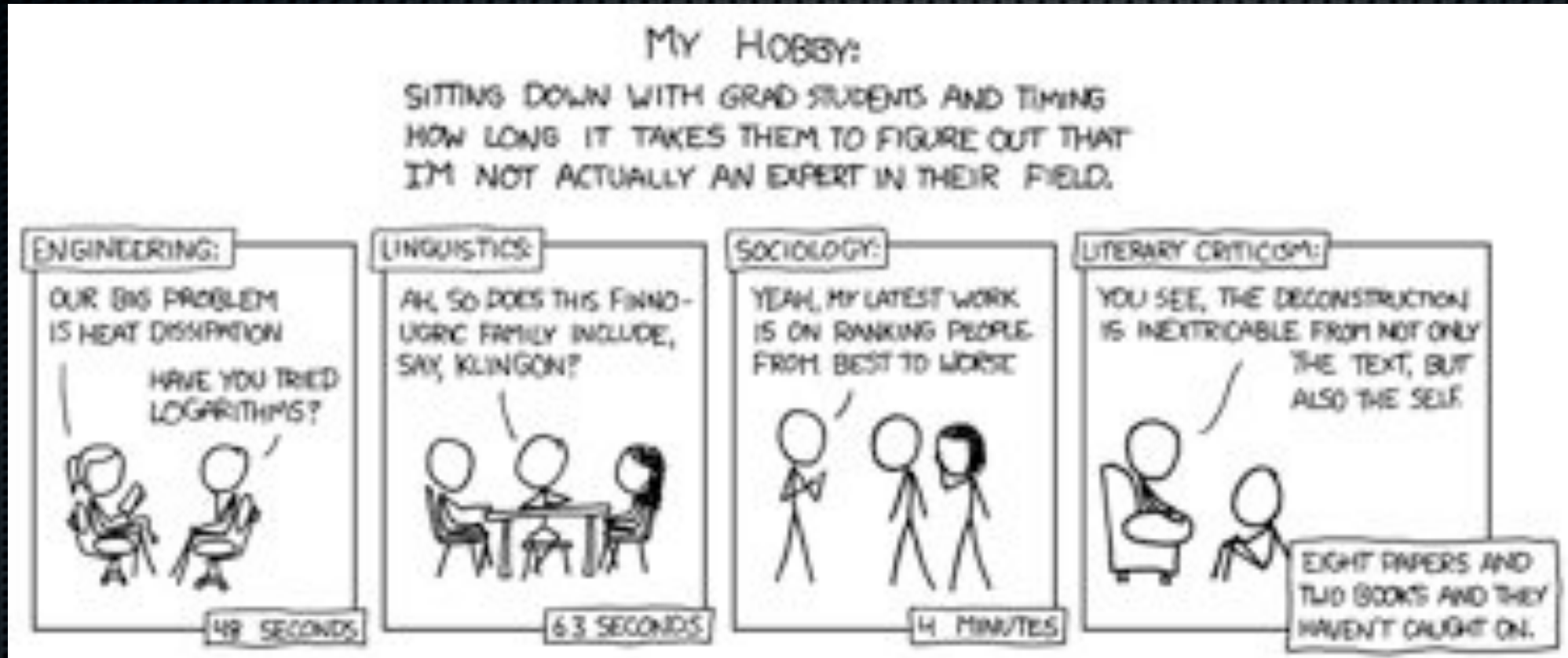
Information Expert Examples

- Who should be responsible for knowing the grand total of a *Sale*?
- Given that a *Piece* just landed on a *Square*, who should be responsible for calculating the rent due?

Information Expert Contraindications

- ✦ Sometimes Information Expert will suggest a solution that leads to coupling or cohesion problems
 - ✦ Consider: Who should be responsible for saving a *Sale* in a database?

Imposter



<http://xkcd.com/451/>

If you think this is too hard on literary criticism, read the Wikipedia article on deconstruction.

Creator

Creator

- **Problem:** Who should be responsible for creating a new instance of some class?
- **Solution:** Make *B* responsible for creating *A* if...
 - *B* contains or is a composition of *A* ← Most important
 - *B* records *A*
 - *B* closely uses *A*
 - *B* has the data to initialize *A*

The more matches
the better.

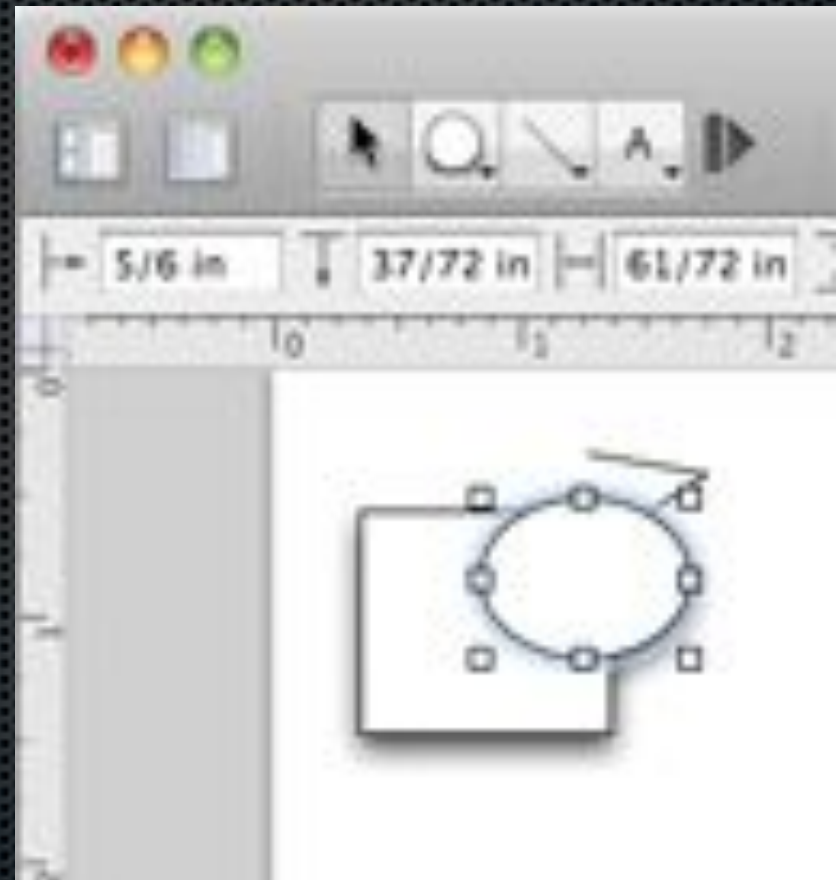
Q7,8

Creator Examples

- ✦ In Monopoly simulator, who should create...
 - ✦ *Squares?*
 - ✦ *Pieces?*
 - ✦ *Dice?*
- ✦ In NextGen POS, who should create...
 - ✦ *SalesLineItems?*
 - ✦ *ProductDescriptions?*

Creator Contraindications

- ✦ Complex creation scenarios
 - ✦ Recycling instances
 - ✦ Conditional creation



Team Creativity

Q11

Controller

Controller

What's that?



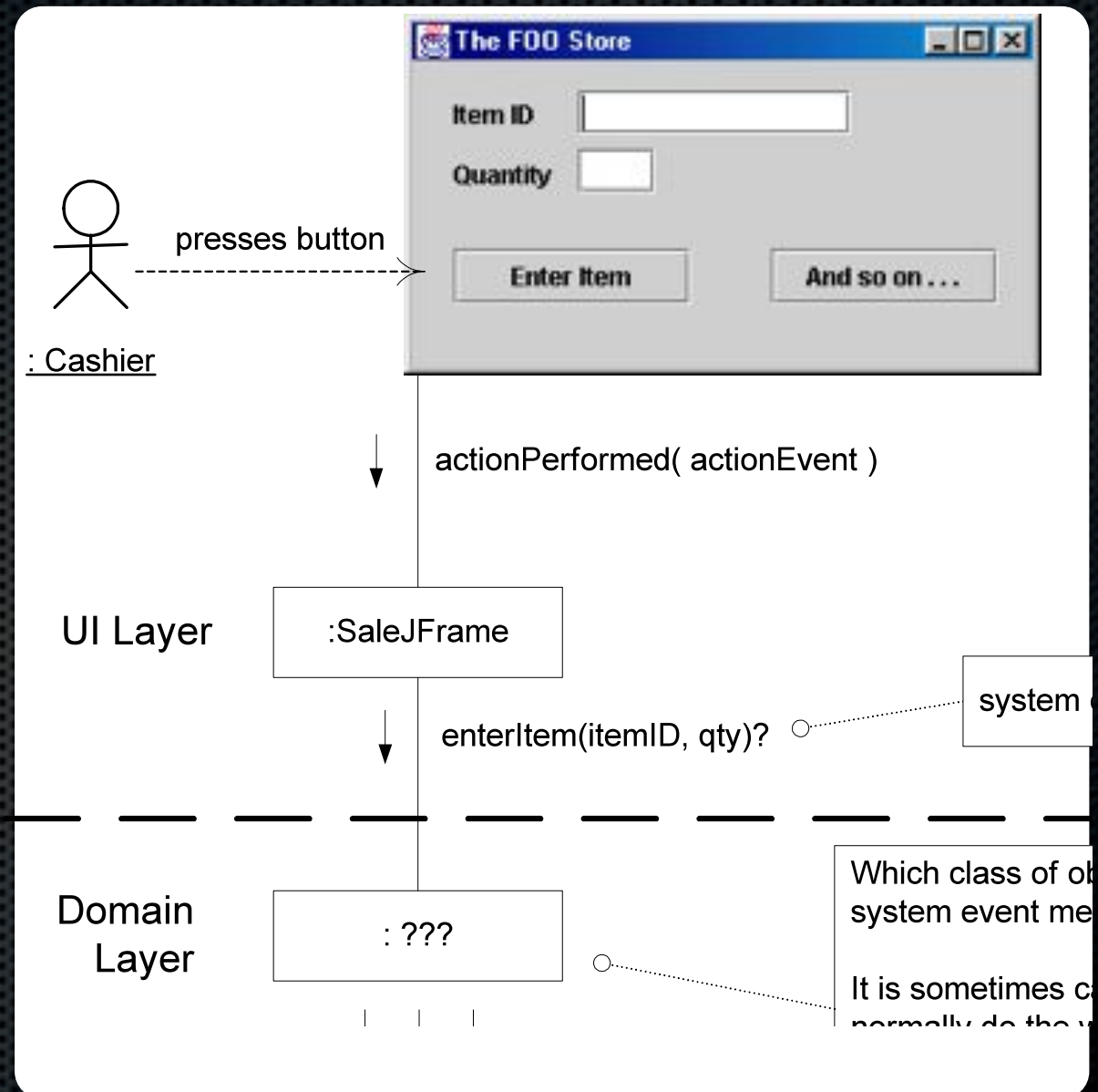
- **Problem:** What first object beyond the UI layer receives and coordinates a **system operation**
- **Solution:** Assign the responsibility to either...
 - A façade controller, representing the overall system and handling all system operations, or
 - A use case controller, that handles all system events for a single use case

Not JFrame, not JPanel, ...

Q12,13

Example

- What domain layer class should own handling of the *enterItem* system operation?



Guidelines

- ✦ Controller should **delegate** to other domain layer objects
- ✦ Use façade controller when...
 - ✦ There are a limited number of system operations, or
 - ✦ When operations are coming in over a single “pipe”
- ✦ Use use case controller when a façade would be bloated (low cohesion!)

Controller Benefits

- ✦ Increased potential for reuse
- ✦ Can reason/control the state of a use case
 - ✦ E.g., don't close sale until payment is accepted

Controller Issues

Switch from façade to
use case controllers

```
graph TD; A[Switch from façade to use case controllers] --> B[Controller bloat—too many system operations]; A --> C[Controller fails to delegate tasks]; A --> D[Controller has many attributes]; E[Delegate!] --> B; E --> C; E --> D;
```

- Controller bloat—too many system operations
- Controller fails to delegate tasks
- Controller has many attributes

Delegate!

Team Control

Q14,15