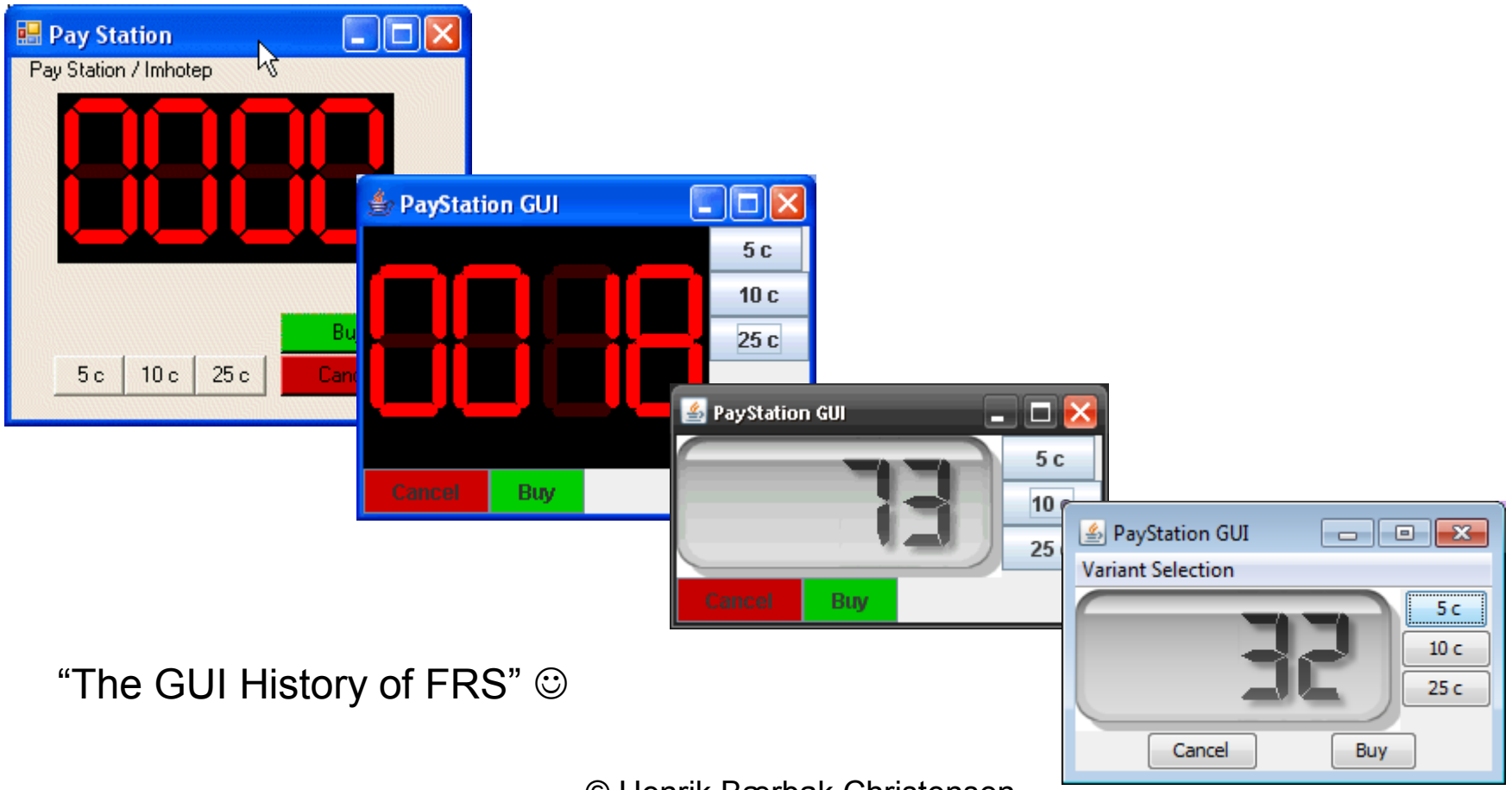
Applying the Principles

Two Examples

Example 1

New Requirement

It would be nice with a simple GUI “to see something” instead of just xUnit tests...



“The GUI History of FRS” ☺

[Demo]

Run it

- Ant gui

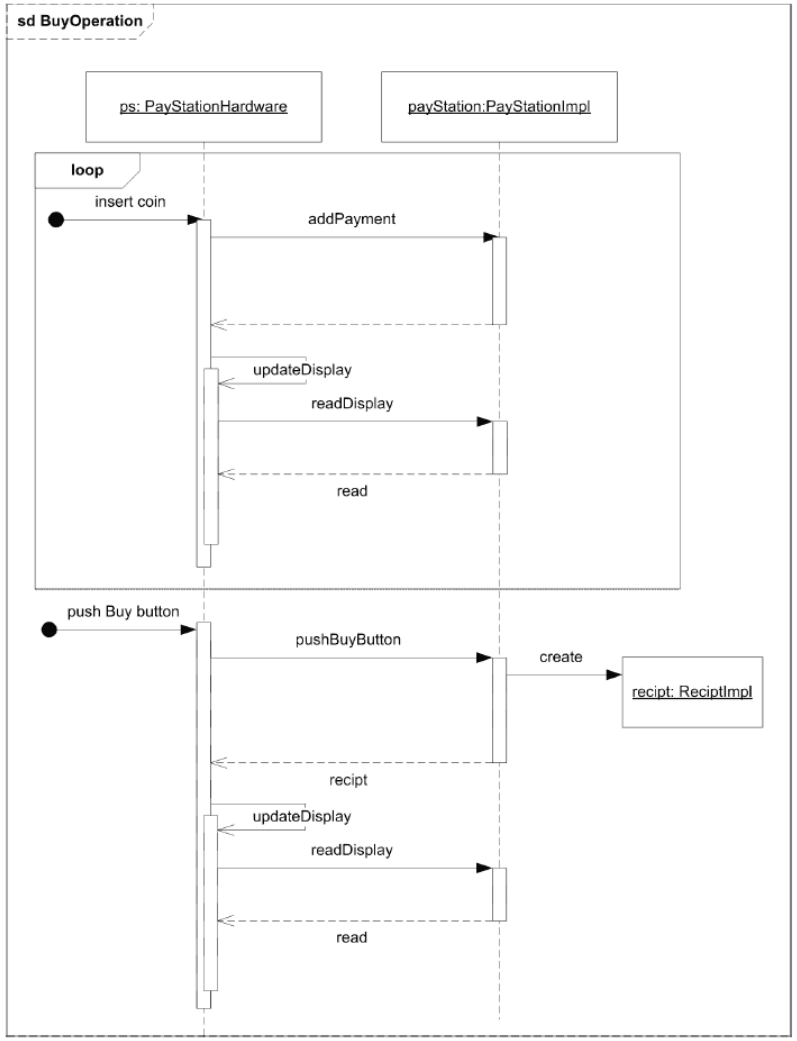
```
/** Create the panel of buttons */
private JComponent createButtonPanel() {
  Box p = new Box( BoxLayout.X_AXIS );
  JButton b;
  b = new JButton("Cancel");
  b.setAlignmentX(Component.CENTER_ALIGNMENT);
  p.add( Box.createHorizontalGlue() );
  p.add( b );
  b.addActionListener( new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        payStation.cancel();
        updateDisplay();
      } } );

  b = new JButton("Buy");
  b.setAlignmentX(Component.CENTER_ALIGNMENT);
  p.add( Box.createHorizontalGlue() );
  p.add( b );
  p.add( Box.createHorizontalGlue() );
  b.addActionListener( new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        Receipt r = payStation.buy();
        updateDisplay();
        // print the receipt
        showReceiptInWindow(r);
      } } );

  return p;
}
```

```
/** Update the digital display with whatever the
    pay station domain shows */
private void updateDisplay() {
  String prefixedZeros =
    String.format("%4d", payStation.readDisplay() );
  display.set( prefixedZeros );
}

/** Create the coin input panel */
private JComponent createCoinInputPanel() {
  Box p = new Box( BoxLayout.Y_AXIS );
  p.add( defineButton( " 5 c", "5" ));
  p.add( defineButton( "10 c", "10" ));
  p.add( defineButton( "25 c", "25" ));
  return p;
}

/** The button action listener that reacts on clicking the
    coin buttons */
private ActionListener buttonActionListener = new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      String s = e.getActionCommand();
      int coin = Integer.parseInt(s);
      try {
        payStation.addPayment( coin );
      } catch (IllegalCoinException exc) {
        // illegal coins just do nothing.
      }
      updateDisplay();
    }
  };
```

Analysis

Seq Diagram

- No difference in behaviour of a GUI versus real hardware!



Conclusion

Any kind of user interface can operate the PayStation!

Wow – *Change by addition...*

How come we are so lucky?

Design considerations

(3) Behaviour that may vary

- the *same* hardware must operate *varying* pay station implementations: AlphaTown, BetaTown, EpsilonTown...

(1) Variable behaviour behind *interface*

- **PayStation** interface...

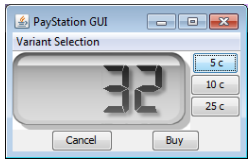
(2) Compose behaviour by delegation

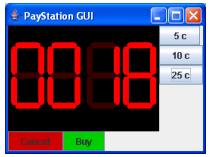
- Gui/Hardware does not itself calculate rates, issue receipts, etc., but *lets an instance of PayStation do the dirty job...*

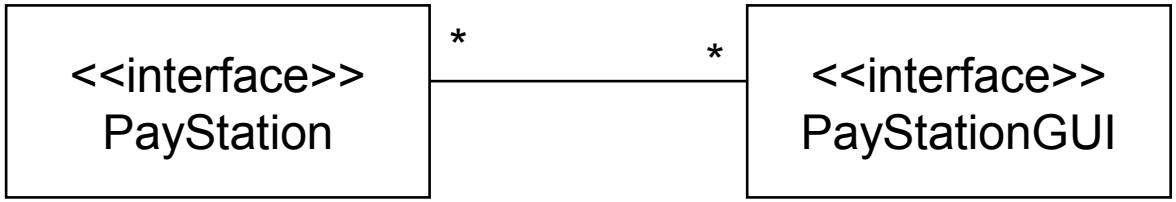
Result

The side effect of this decisions is that *interface decouples both ways!!!*

- *Hardware may operate different kinds of PayStation implementations*
  - *Alpha, Beta, Gamma, …*
- *Different kinds of user interfaces may operate the same PayStation implementation*

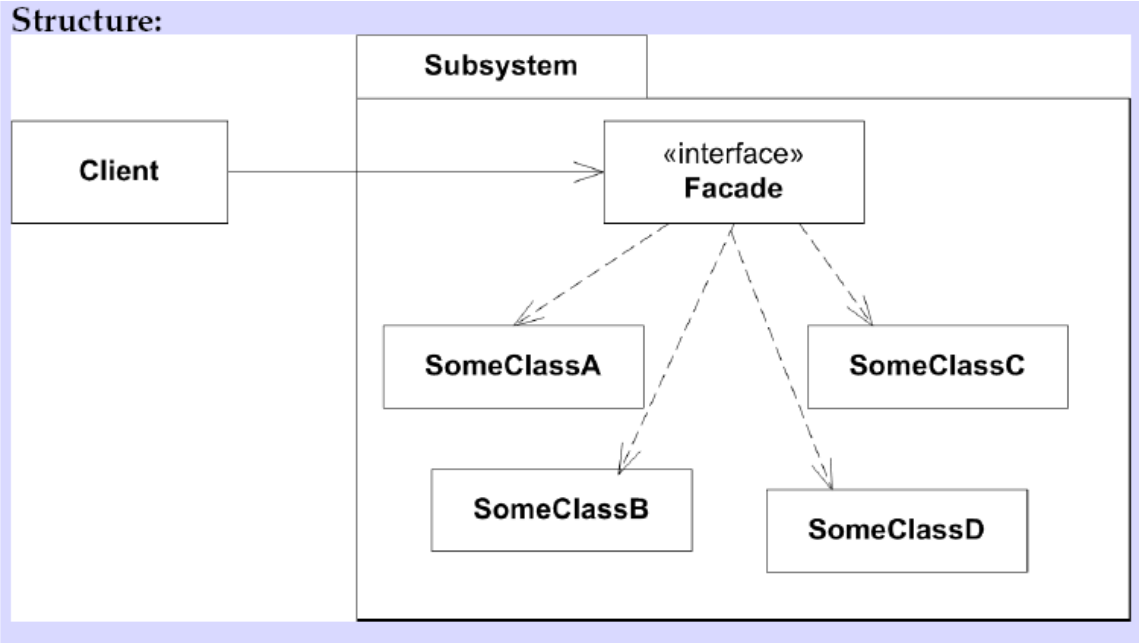






Automagical pattern?

PayStation is an example of the **Facade** pattern



Consequences

Benefits

- Shields clients from subsystem objects
  - (depends… Consider HotCiv)
- Weak coupling
  - **Many to many** relation between client and façade

Liabilities

- Bloated interface with *lots of methods*
  - Because façade must have the sum of responsibilities of the subsystem
- How to avoid access to the inner objects?
  - Read-only interfaces; no access (require dumb data objects to be passed and parsed over the façade).

Example 2

New Requirement

Alphatown wants to log all coin entries:

- [time] [value]

Example:

- 14:05:12 5 cent
- 14:05:14 25 cent
- 14:55:10 25 cent

The 3-1-2 machinery

Let us look at the machinery:

③ Identify the responsibility whose concrete behaviour may vary

① Express responsibility as an interface

② Let someone else do the job

How does this apply?

What is 3-1-2 here?

Analysis

③ Identify the responsibility whose concrete behaviour may vary

- It is the “Accept payment” responsibility

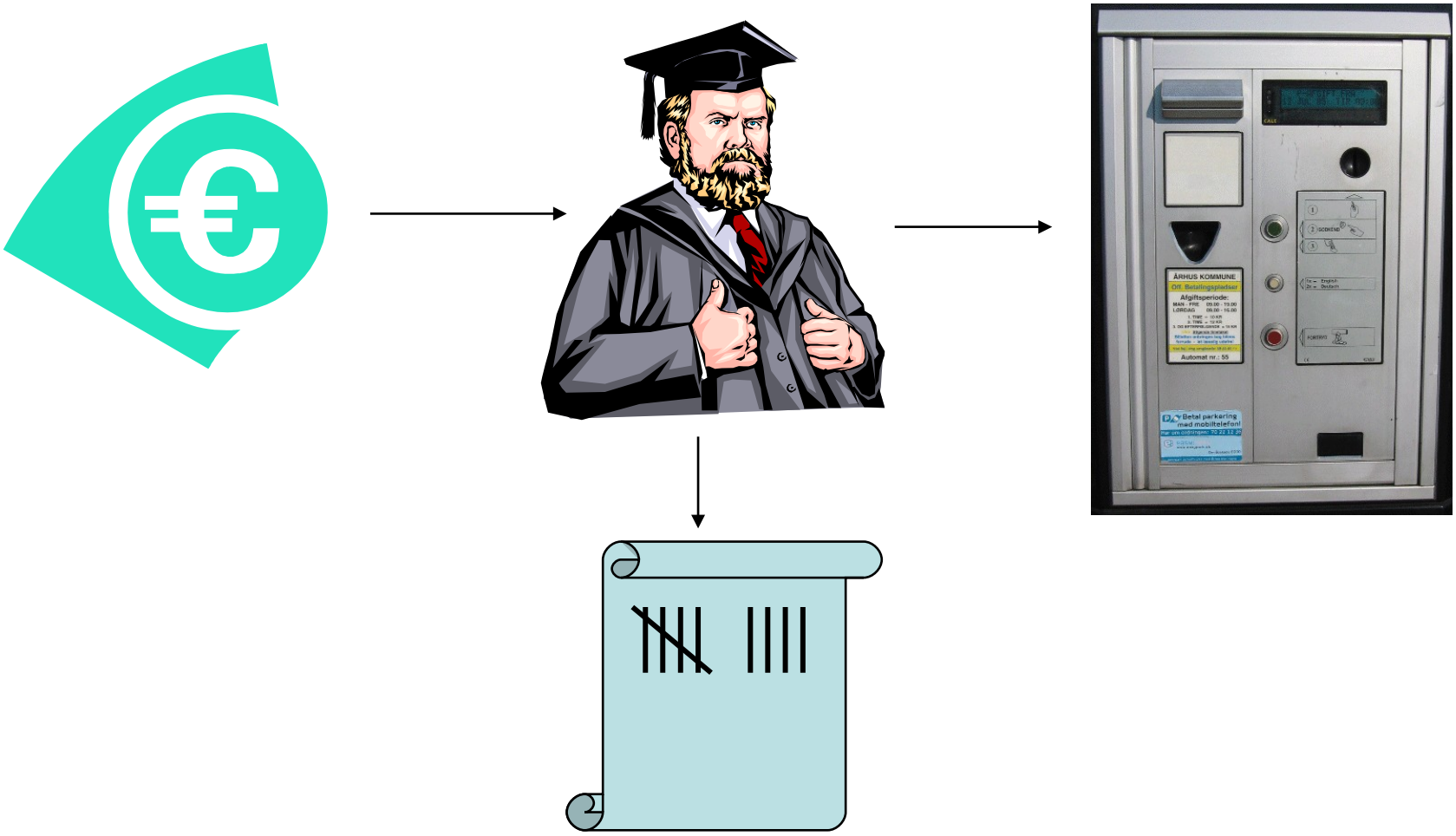
① Express responsibility as an interface

- A) PaymentAccepter role?
- B) PayStation role? Already in place!

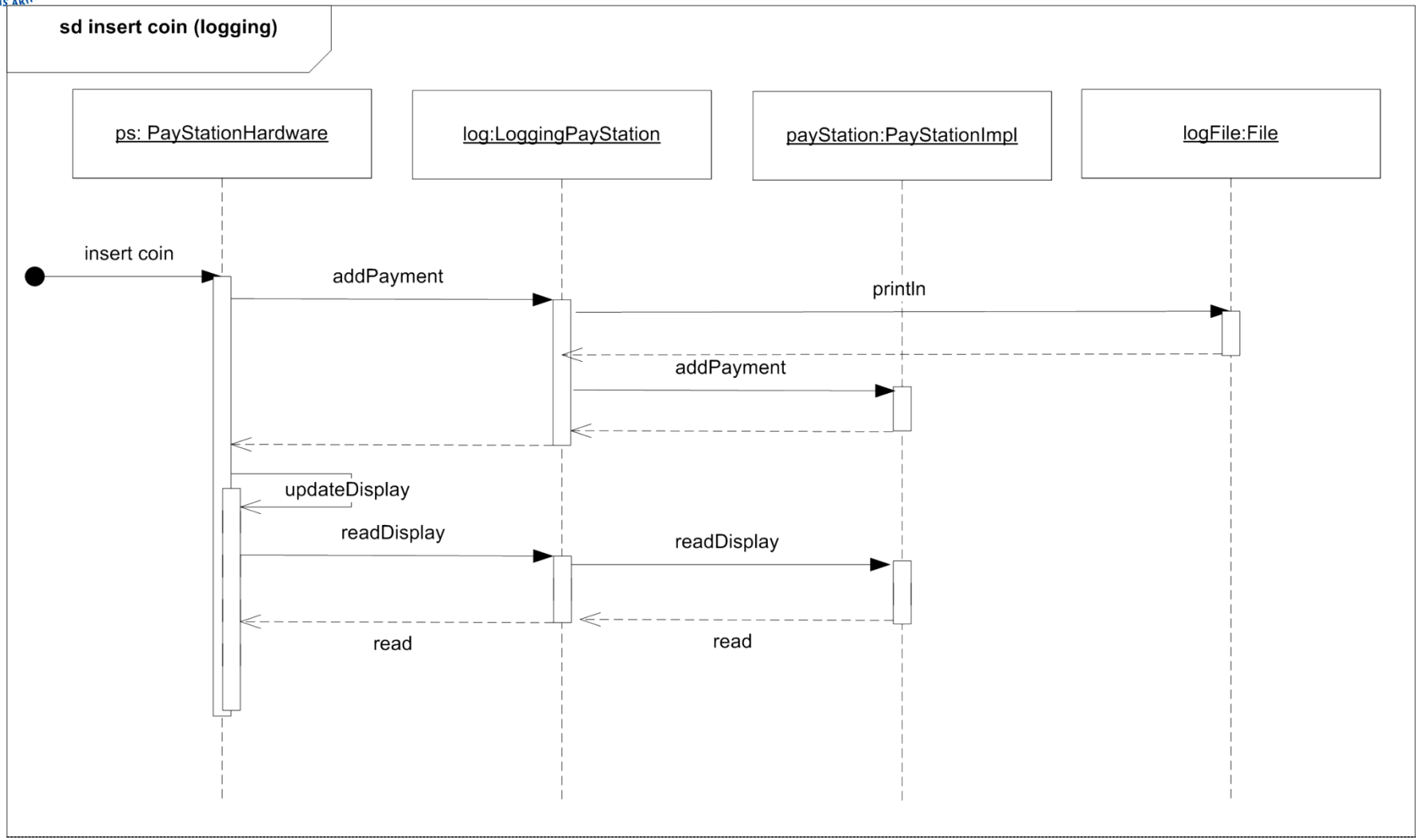
② Let someone else do the job

- Maybe let someone handle the coins *before* the parking machine receives them?

Metaphor: Principle 2



Dynamics

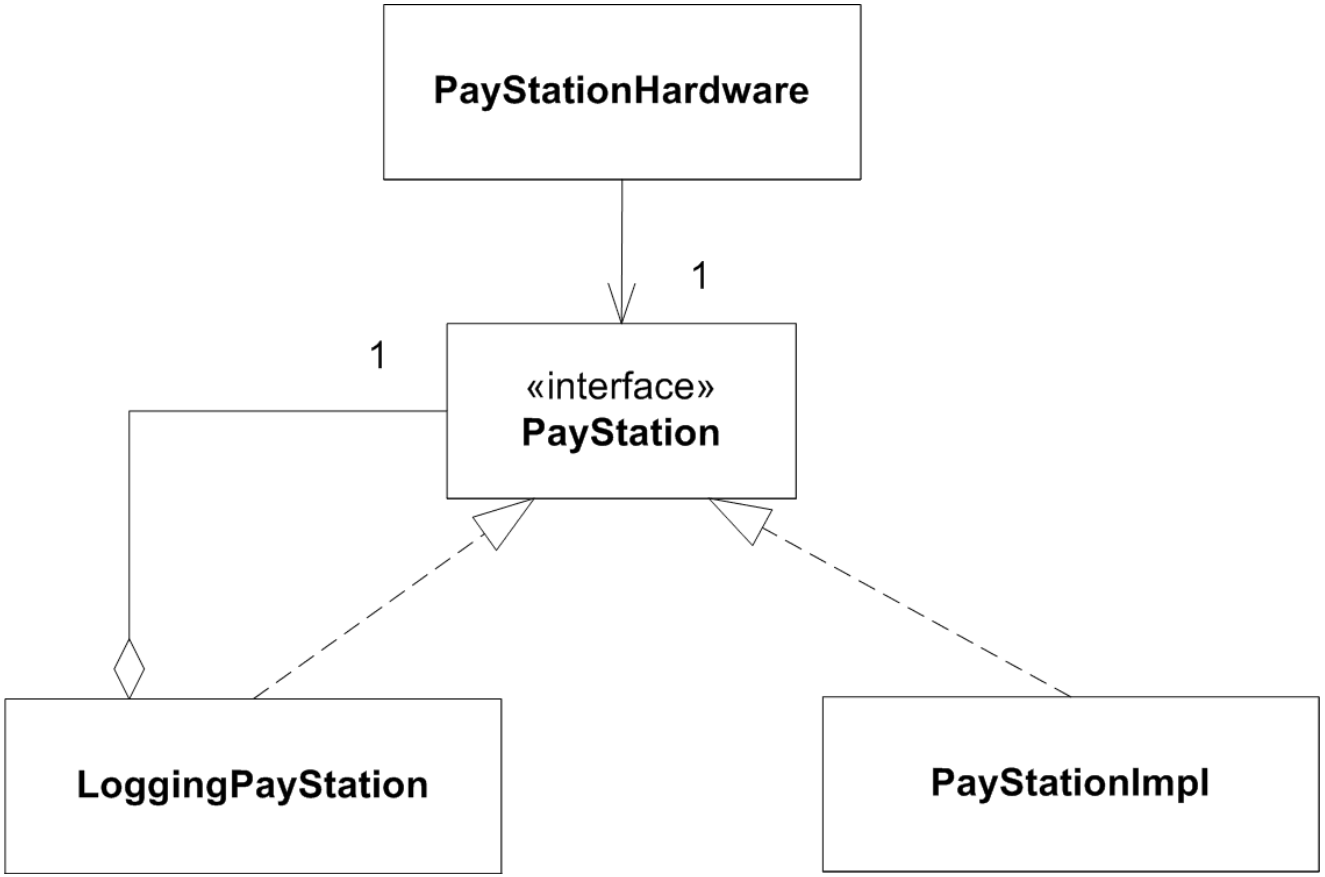

sd insert coin (logging)
ps: PayStationHardware
log:LoggingPayStation
payStation:PayStationImpl
logFile:File
insert coin
addPayment
println
addPayment
updateDisplay
readDisplay
readDisplay
read
read

[Demo]

Refactoring process

- Introduce Null Decorator
- Run, see everything pass
- Introduce feature in decorator

Statics

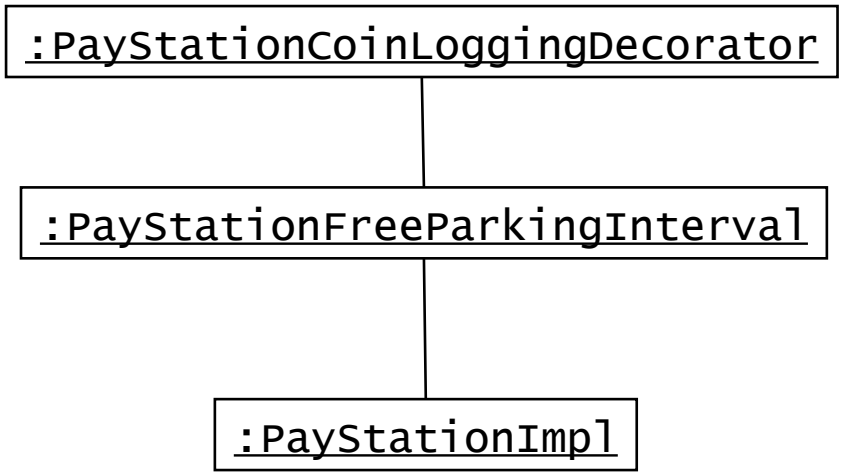

PayStationHardware
1
1
«interface»
PayStation
LoggingPayStation
PayStationImpl

Chaining decorators

Decorators can form chains.

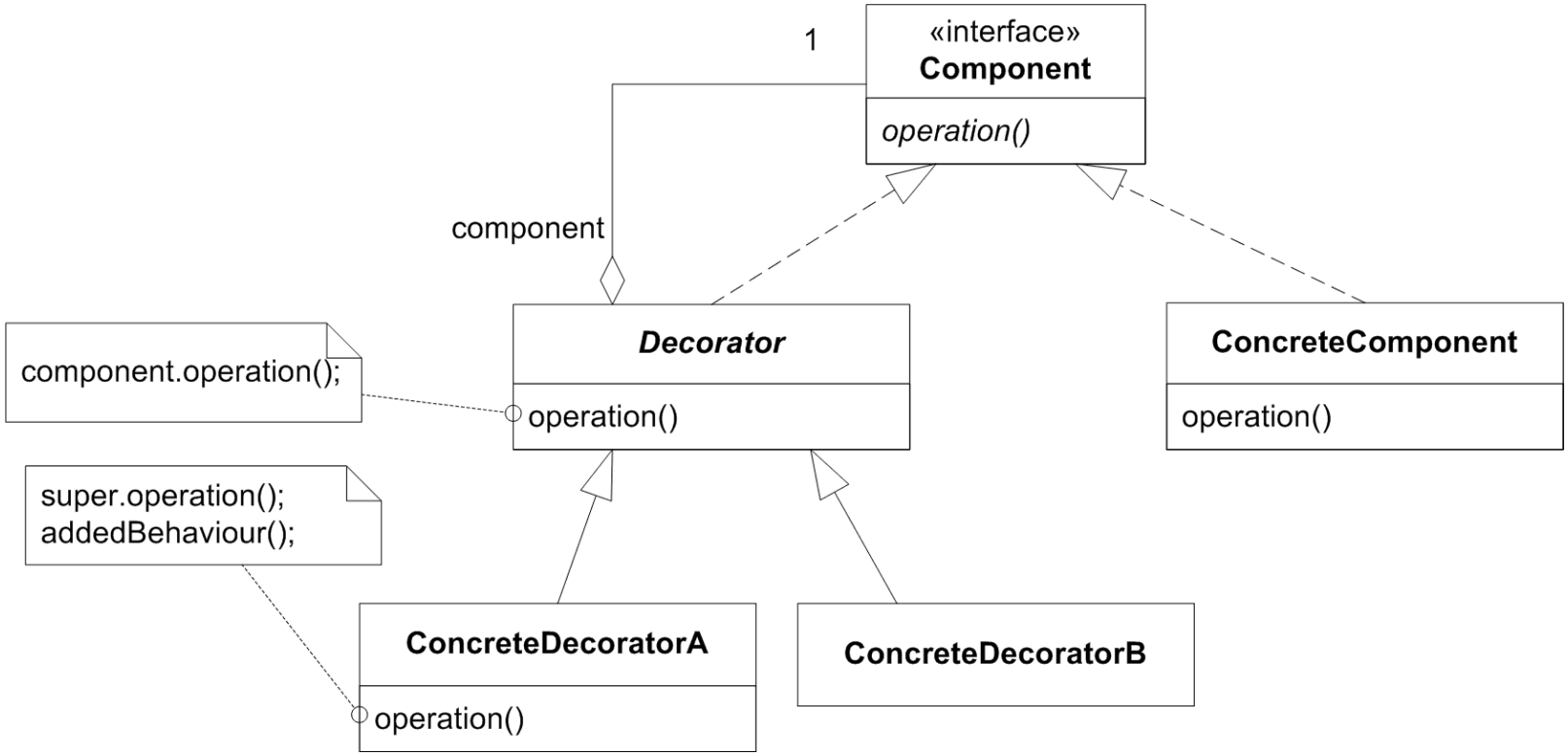
New requirement:

- no payment possible in 19.00 – 07.00 interval



Automagical pattern?

The decorator is yet another application of 3-1-2 and the principles of flexible design!



Consequences

Benefits

- Adding and removing behavior at run-time
- Incrementally add responsibilities
- Complex behavior by chaining decorators

Liabilities

- Analyzability suffers as you end up with lots of little objects
  - Behavior is constructed at run-time instead of being written in the static code
- Delegation code tedious to write
  - Make a ‘null decorator’ as base class