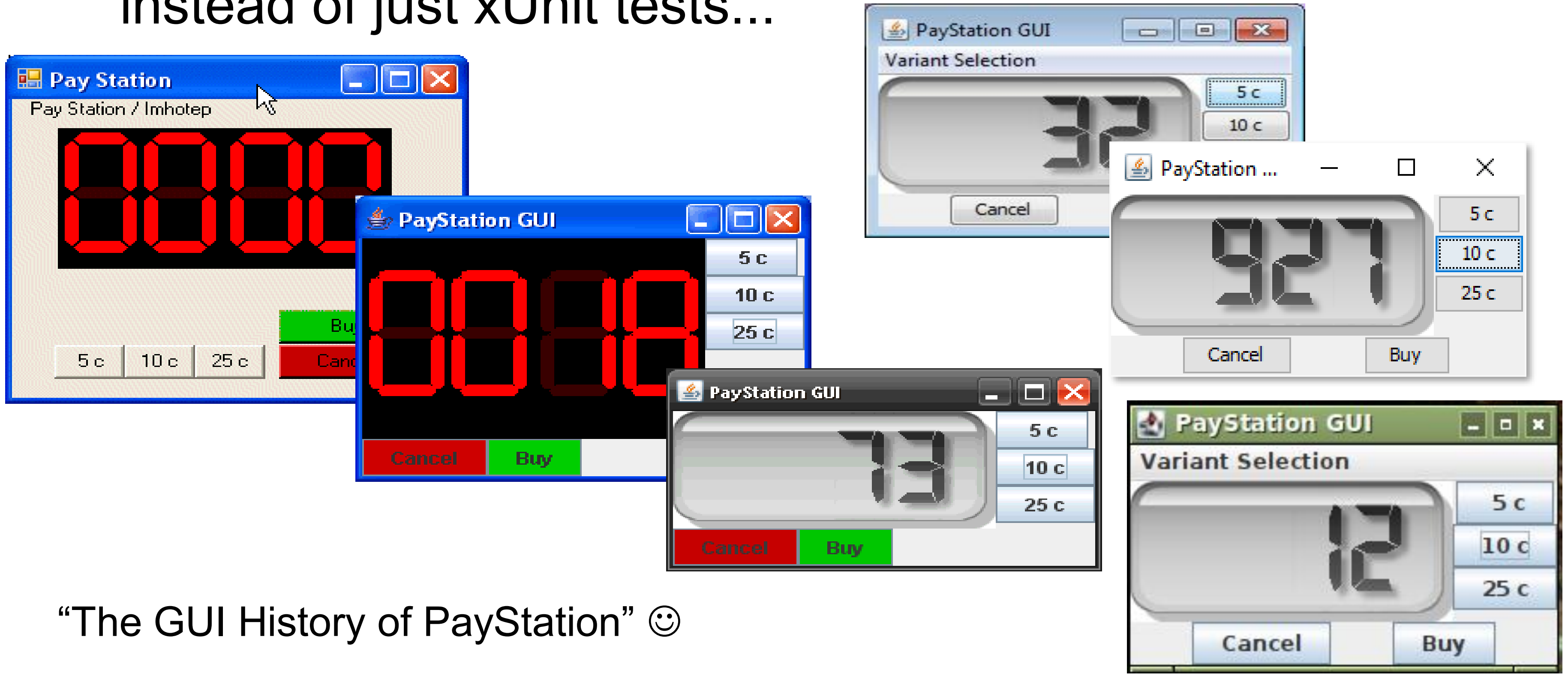
AARHUS UNIVERSITET

Software Engineering and Architecture

Pattern Catalog: Façade

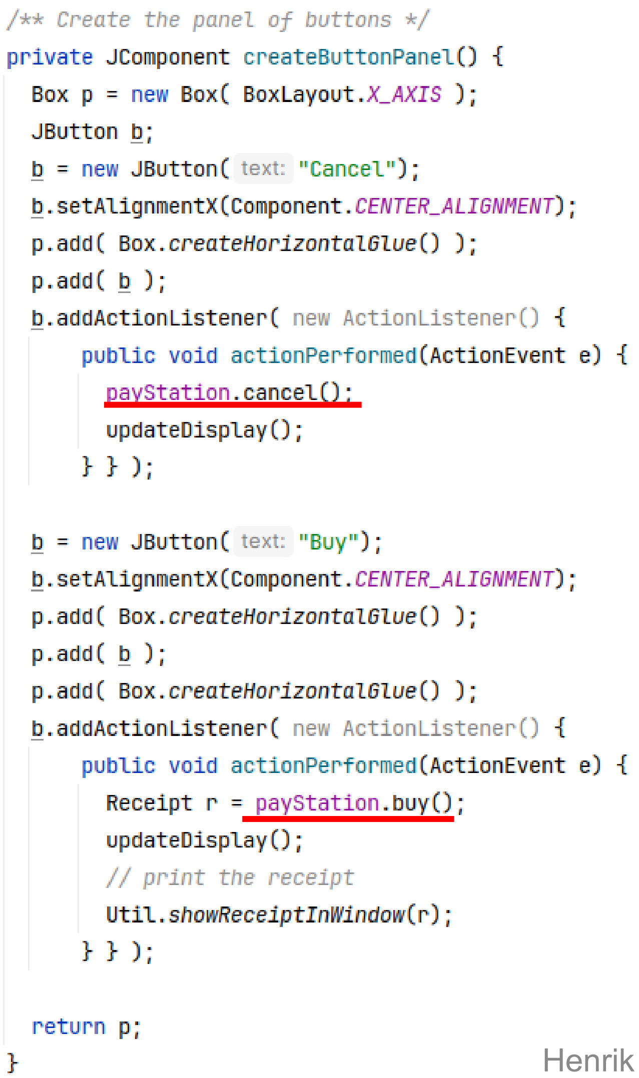
New Requirement

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



“The GUI History of PayStation” ☺

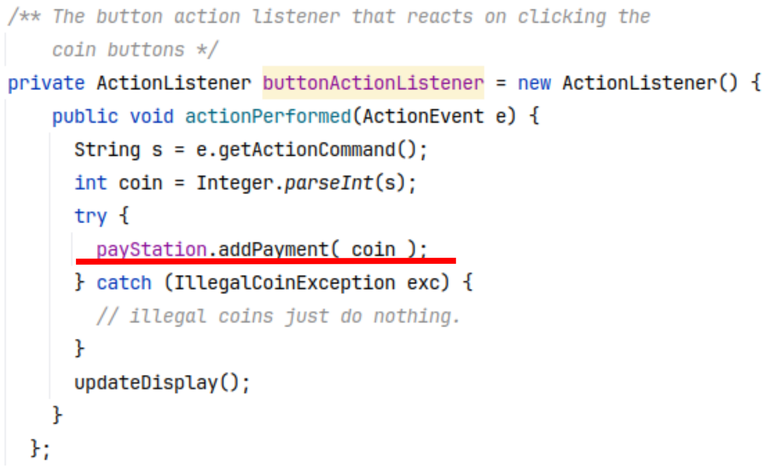
[Demo]



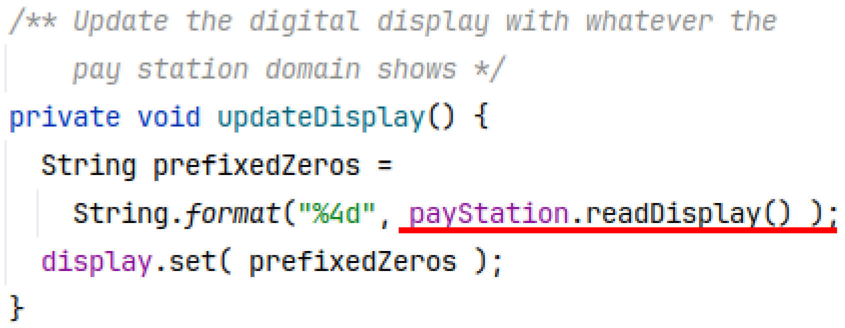
```java
/** Create the panel of buttons */
private JComponent createButtonPanel() {
  Box p = new Box( BoxLayout.X_AXIS );
  JButton b;
  b = new JButton( text: "Cancel");
  b.setAlignmentX(Component.CENTER_ALIGNMENT);
  p.add( Box.createHorizontalGlue() );
  p.add( b );
  b.addActionListener( new ActionListener()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

  b = new JButton( text: "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
        Util.showReceiptInWindow(r);
      } } );

  return p;
}
```



```java
/** 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();
    }
  };
```

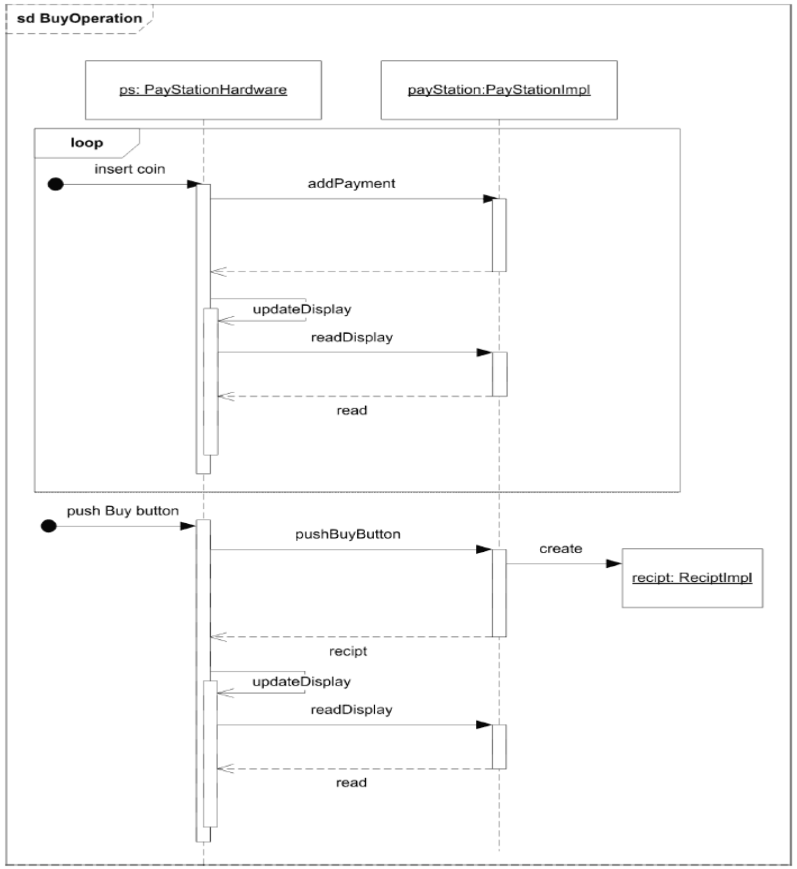


```java
/** Update the digital display with whatever the
    pay station domain shows */
private void updateDisplay() {
  String prefixedZeros =
    String.format("%4d", payStation.readDisplay() );
  display.set( prefixedZeros );
}
```

AARHUS UNIVERSITET

Analysis

- Seq Diagram
  - No difference in behavior 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?

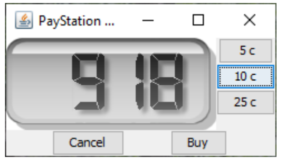
AARHUS UNIVERSITET

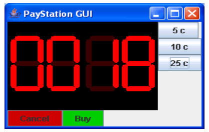
Design considerations

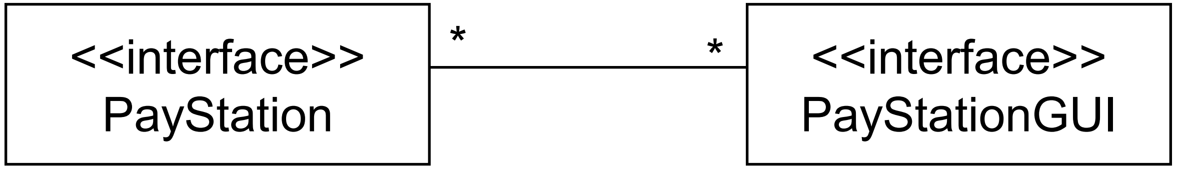
- ③ Behavior that may vary
  - the *same* hardware (or GUI) must operate *varying* pay station implementations: AlphaTown, BetaTown, EpsilonTown...
- ① Variable behavior behind *interface*
  - **PayStation** interface...
- ② Compose behavior by delegation
  - Gui/Hardware does not itself calculate rates, issue receipts, etc., but *lets an instance of PayStation do the dirty job...*

Result

- The side effect of this decision is that *interface decouples both ways!!!*
  - *User interfaces may operate different kinds of PayStation implementations*
    - *Alpha, Beta, Gamma, …*
  - *Different kinds of user interfaces may operate the same PayStation implementation*



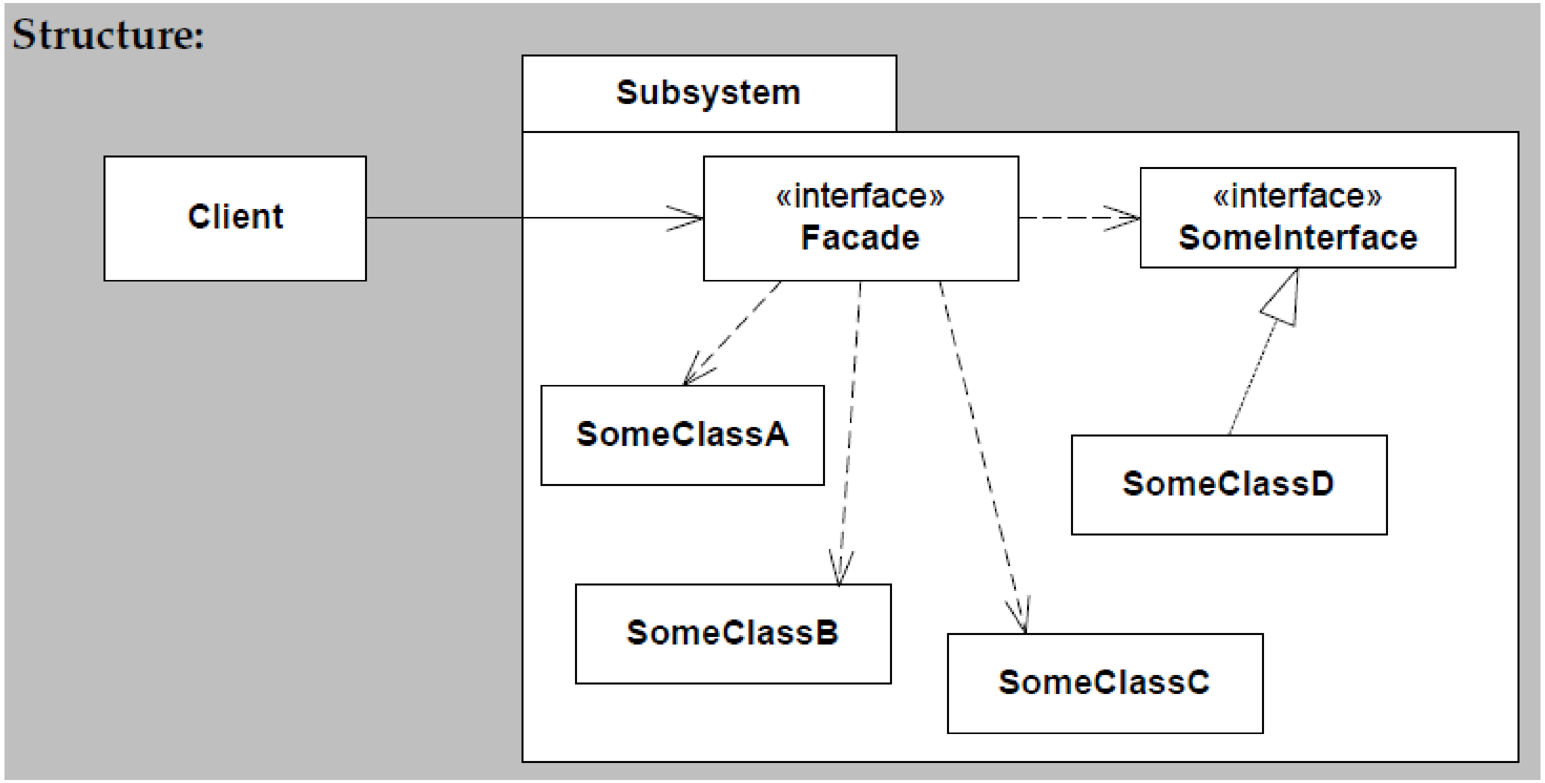




AARHUS UNIVERSITET

Automagical pattern?

- PayStation is an example of the **Façade** pattern



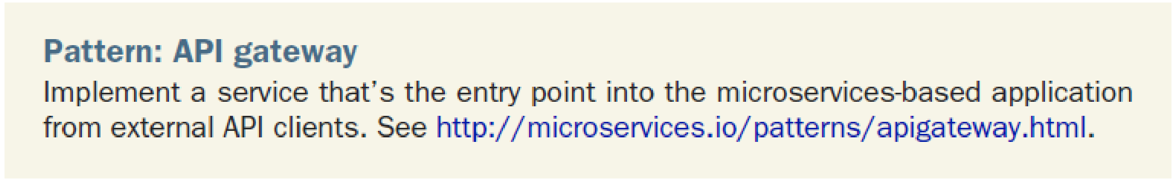
Consequences

- Benefits
  - Shields clients from subsystem objects
    - (depends… Consider HotStone)
  - 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 for “outside”, and *private interfaces* (mutating) on the “inside” of the façade/sub system boundary

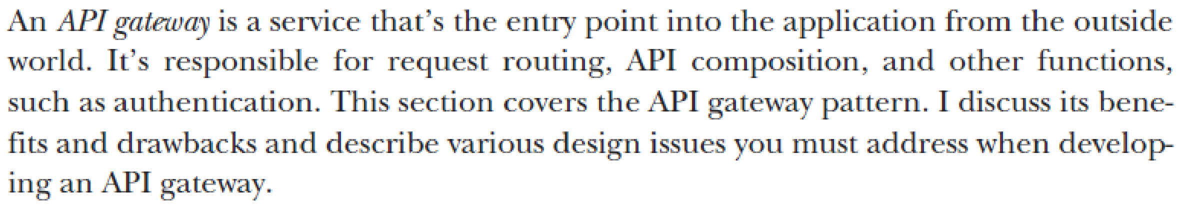
AARHUS UNIVERSITET

Facade Variants

- *Facade appears on all levels of architecture!*
  - An Java Interface is a ‘small scale’ Facade…
  - The *Layered architectural style* defines a set of Facades
  - At remote service level, we have *API Gateway*



Pattern: API gateway
Implement a service that’s the entry point into the microservices-based application from external API clients. See http://microservices.io/patterns/apigateway.html.



An *API gateway* is a service that’s the entry point into the application from the outside world. It’s responsible for request routing, API composition, and other functions, such as authentication. This section covers the API gateway pattern. I discuss its benefits and drawbacks and describe various design issues you must address when developing an API gateway.

Compiler

- A Compiler is a pretty complex piece of software
  - You will learn soon enough ☺ ‘Oversættelse’
  - Lexical analyzer, Syntax Analysis, Parser, Symbol table, Code generator…
- Hidden behind an extremely simple **Façade**
  - ‘javac HelloWorld.java’
    - The façade has one method
      - public ClassFile compile(String filename)