CSSE 374: Even More Object Design with Gang of Four Design Patterns

Shawn Bohner

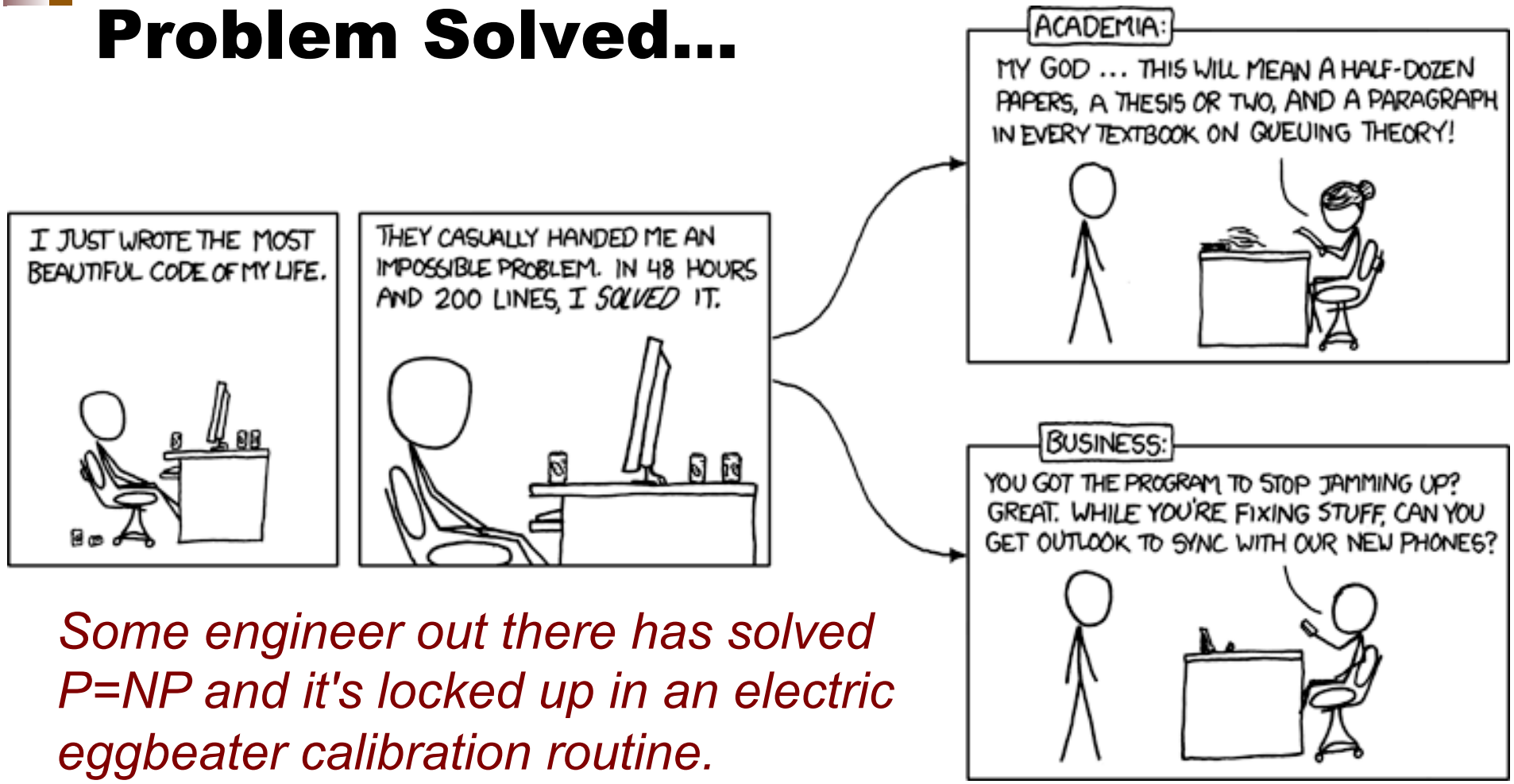
Office: Moench Room F212

Phone: (812) 877-8685

Email: bohner@rose-hulman.edu

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Problem Solved...



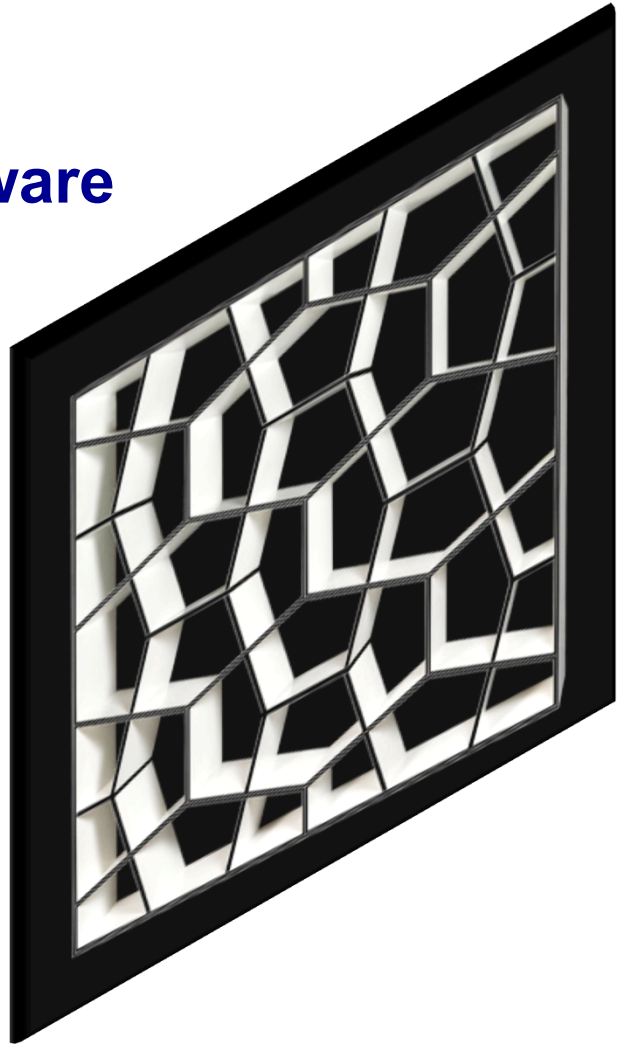
Some engineer out there has solved P=NP and it's locked up in an electric eggbeater calibration routine.

For every 0x5f375a86 we learn about, there are thousands we never see.

Learning Outcomes: Patterns, Tradeoffs

Identify criteria for the design of a software system and select patterns, create frameworks, and partition software to satisfy the inherent trade-offs.

- Using GoF Patterns in Iteration 3
  - Support for third-party POS devices
  - Handling payments
- Design Studio with Team 2.5



Supporting 3rd Party Devices:

How would you handle external, 3rd party devices that have largely the same function, but they might operate differently and have different interfaces?

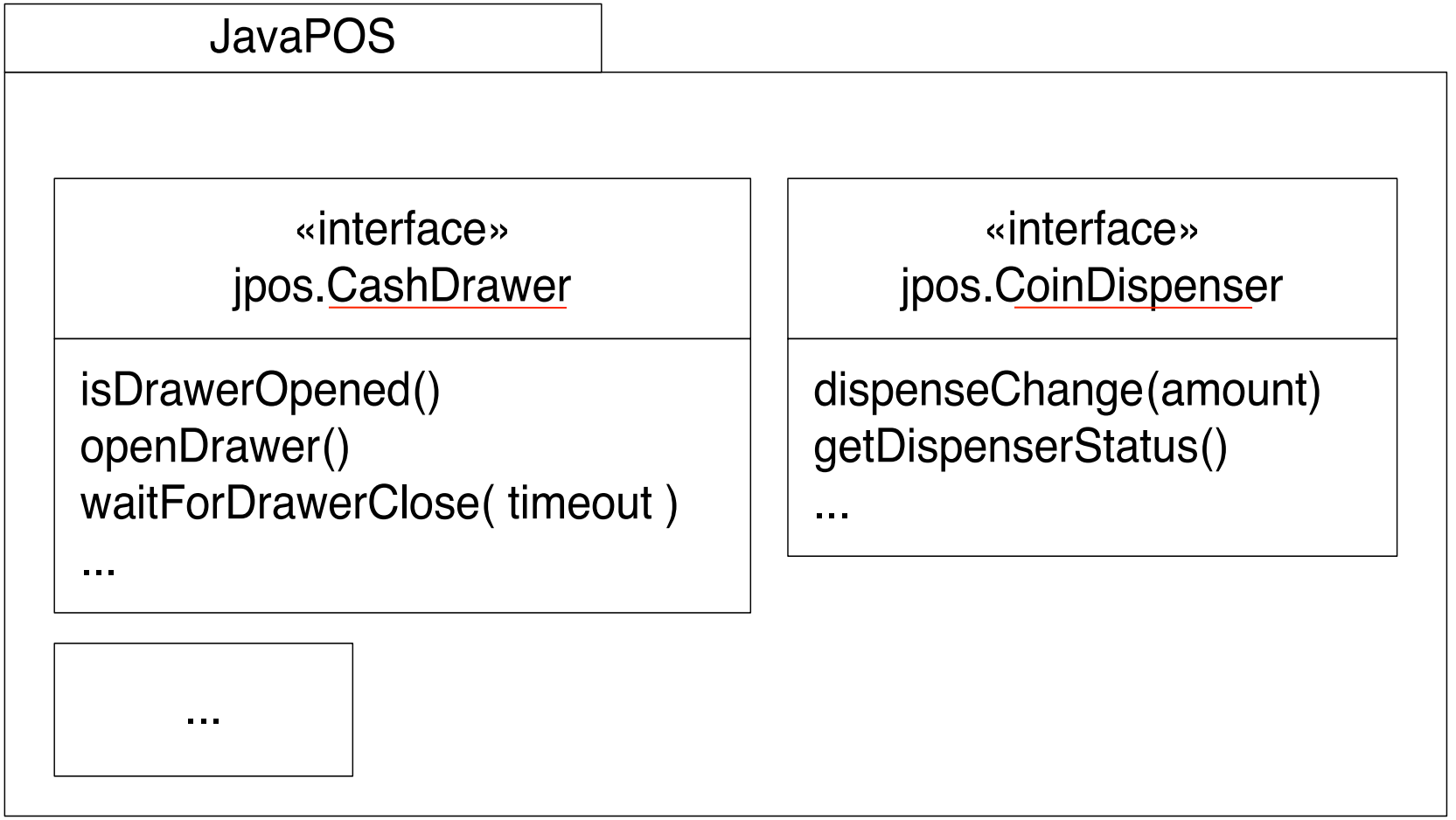
- Think for 15 seconds…
- Turn to a neighbor and discuss it for a minute

Accessing External Physical Devices

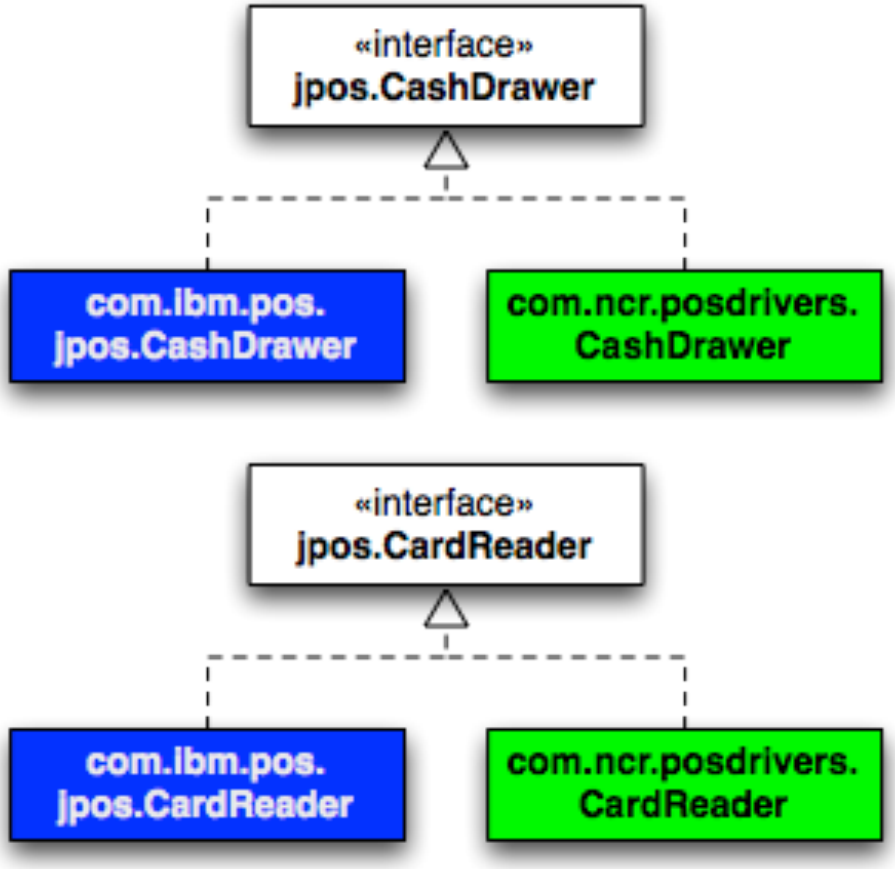
- POS devices include cash drawer, coin dispenser, digital signature pad, & card reader
- They must work with devices from a variety of vendors like IBM, NCR, Fijitsu …

- UnifiedPOS: an industry standard OO interface
  - JavaPOS provides a Java mapping as a set of Java interfaces

Standard JavaPOS Interfaces for Hardware Device Control



Manufacturers Provide Implementations



- Device driver for hardware
- The Java class for implementing JavaPOS interface

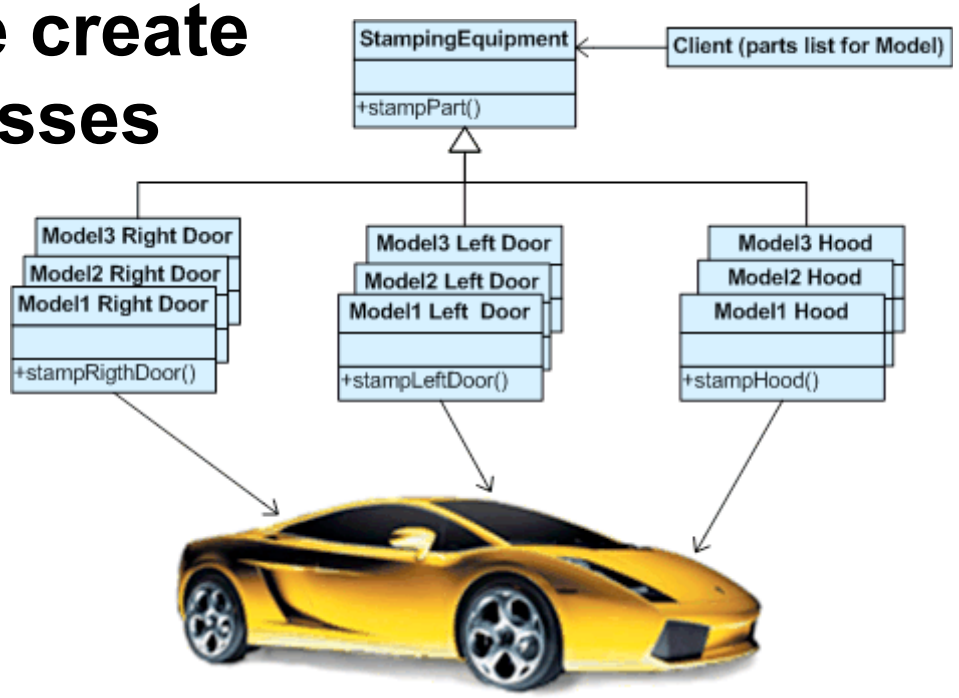
What does this mean for NextGen POS?

- What types does NextGen POS use to communicate with external devices?
- How does NextGen POS get the appropriate instances?

Assume: A given store uses a single manufacturer

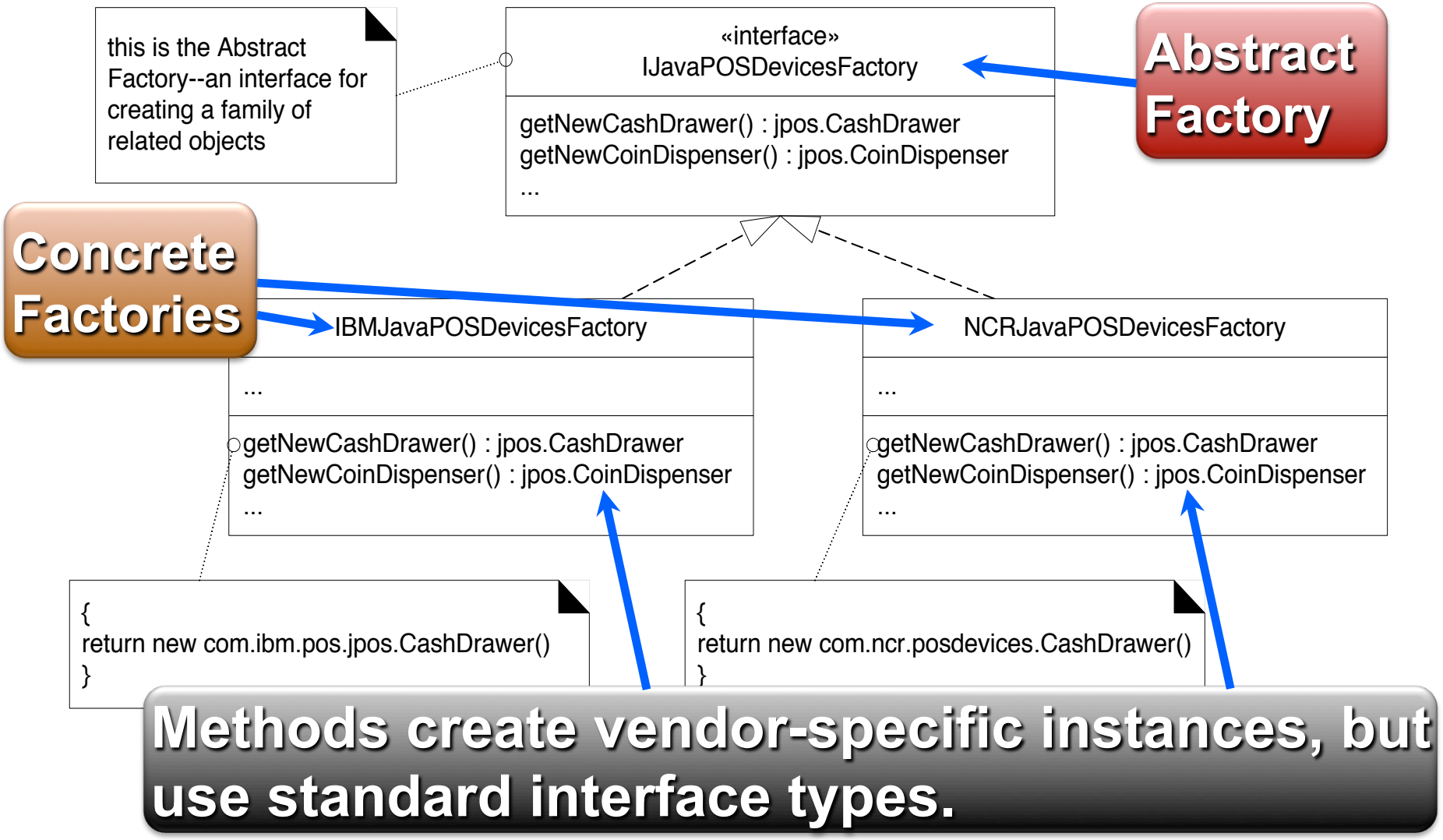
Closer look at Abstract Factory

- **Problem**: How can we create families of related classes while preserving the variation point of switching between families?



- **Solution**:
  Define an *abstract factory* interface.
  Define a *concrete factory* for each family.

Abstract Factory Example



1st Attempt at Using Abstract Factory

```
class Register {
        ...
        public Register() {
                IJavaPOSDevicesFactory factory =
                        new IBMJavaPOSDevicesFactory();
                this.cashDrawer =
                        factory.getNewCashDrawer();
                ...
        }
}
```

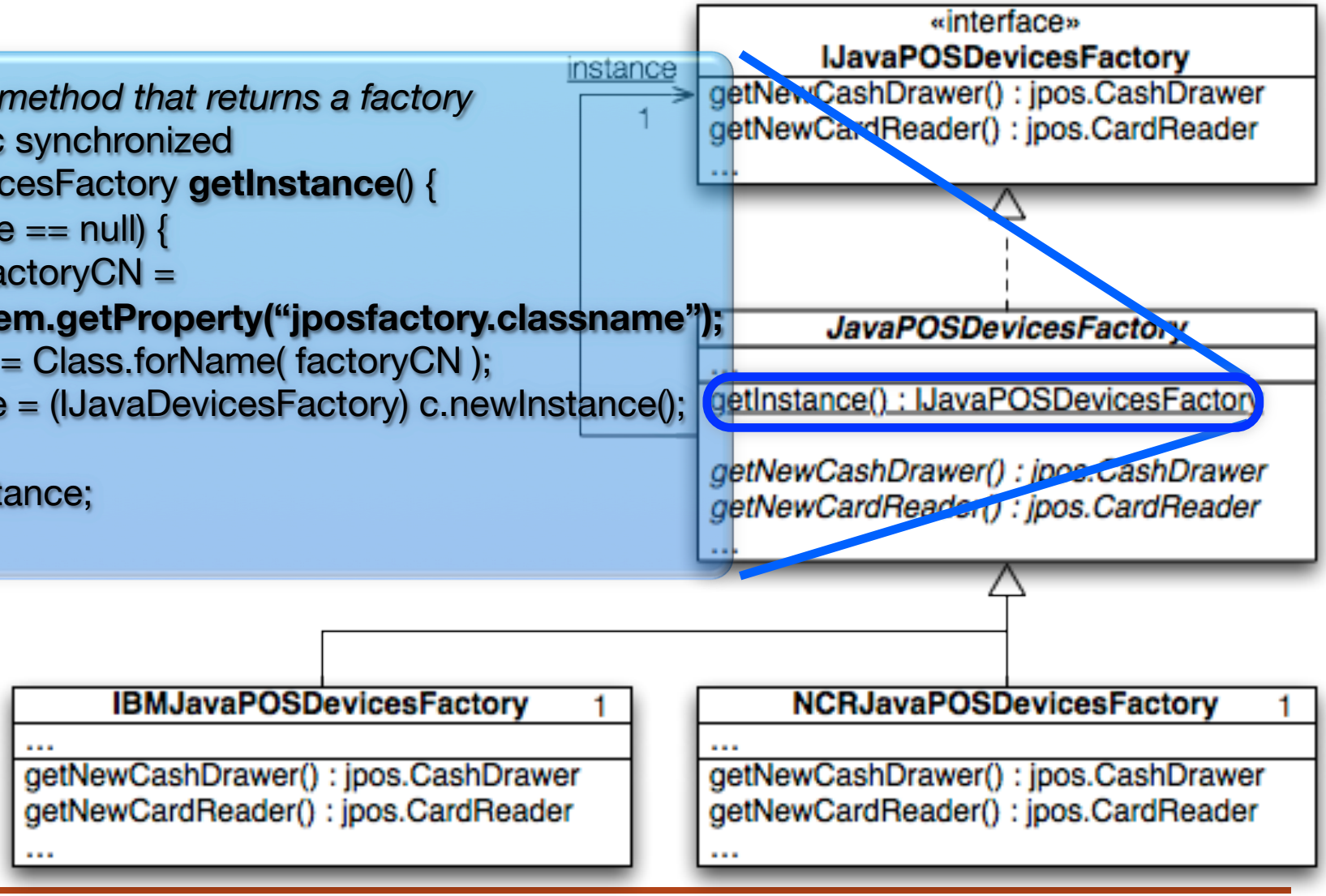
Constructs a vendor-specific concrete factory

Uses it to construct device instances

What if we want to change vendors?

Use an Abstract Class Abstract Factory

```
// A factory method that returns a factory
public static synchronized
   IJavaDevicesFactory getInstance() {
   if (instance == null) {
      String factoryCN =
          System.getProperty("jposfactory.classname");
      Class c = Class.forName( factoryCN );
      instance = (IJavaDevicesFactory) c.newInstance();
   }
   return instance;
}
```



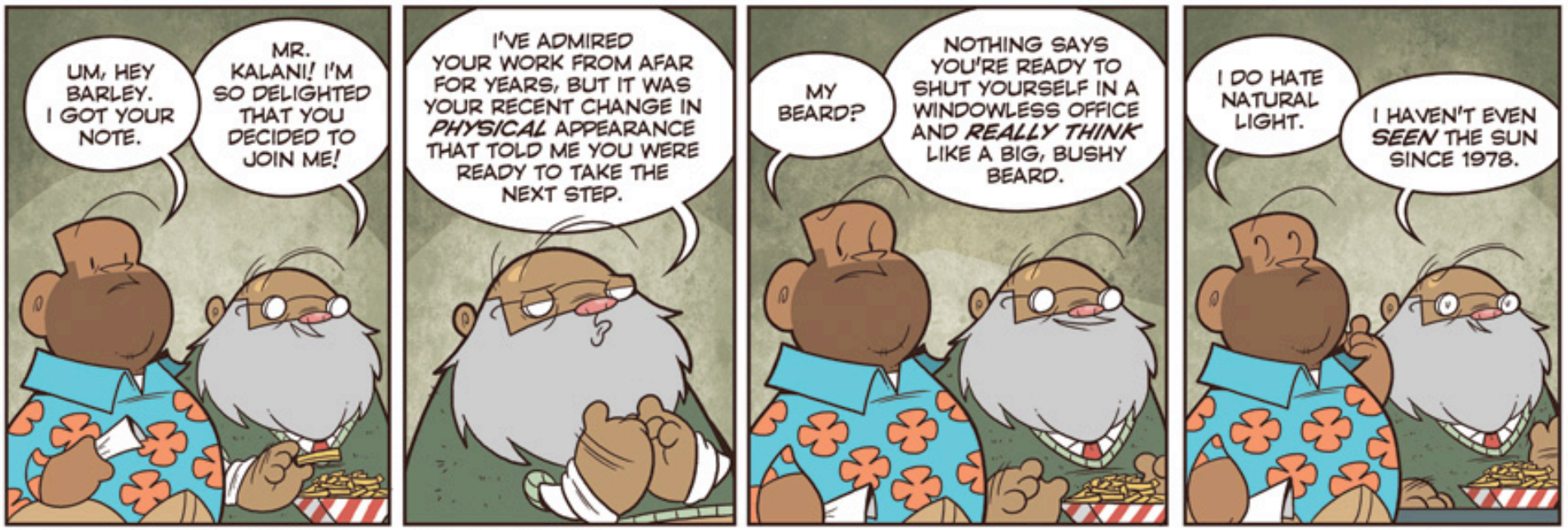
Using a Factory Factory

```
class Register {
        ...
        public Register() {
                IJavaPOSDevicesFactory factory =
                        JavaPOSDevicesFactory.getInstance();
                this.cashDrawer =
                        factory.getNewCashDrawer();
                ...
        }
}
```

Gets a vendor-specific concrete factory singleton

Uses it to construct device instances

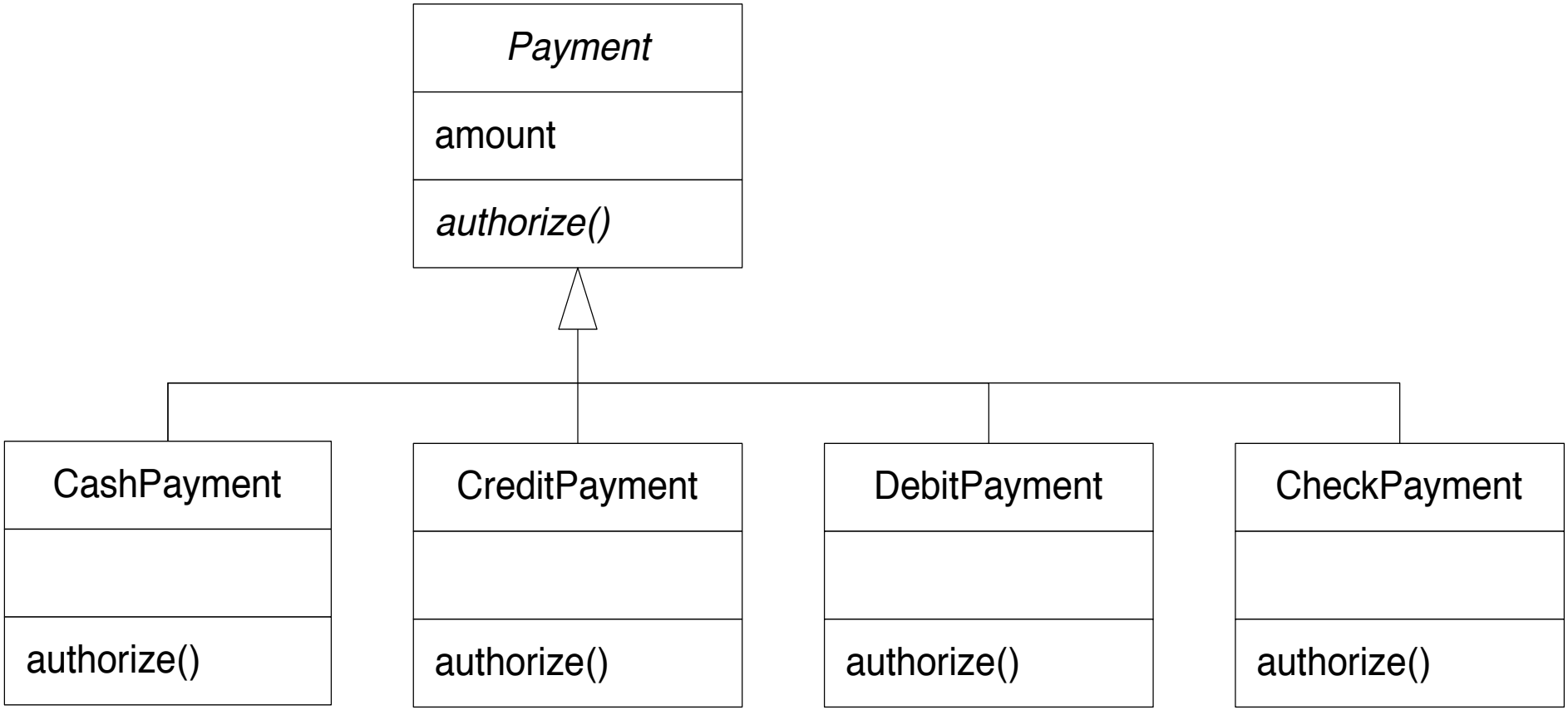
Politics in the Software Organization



Handling Payments

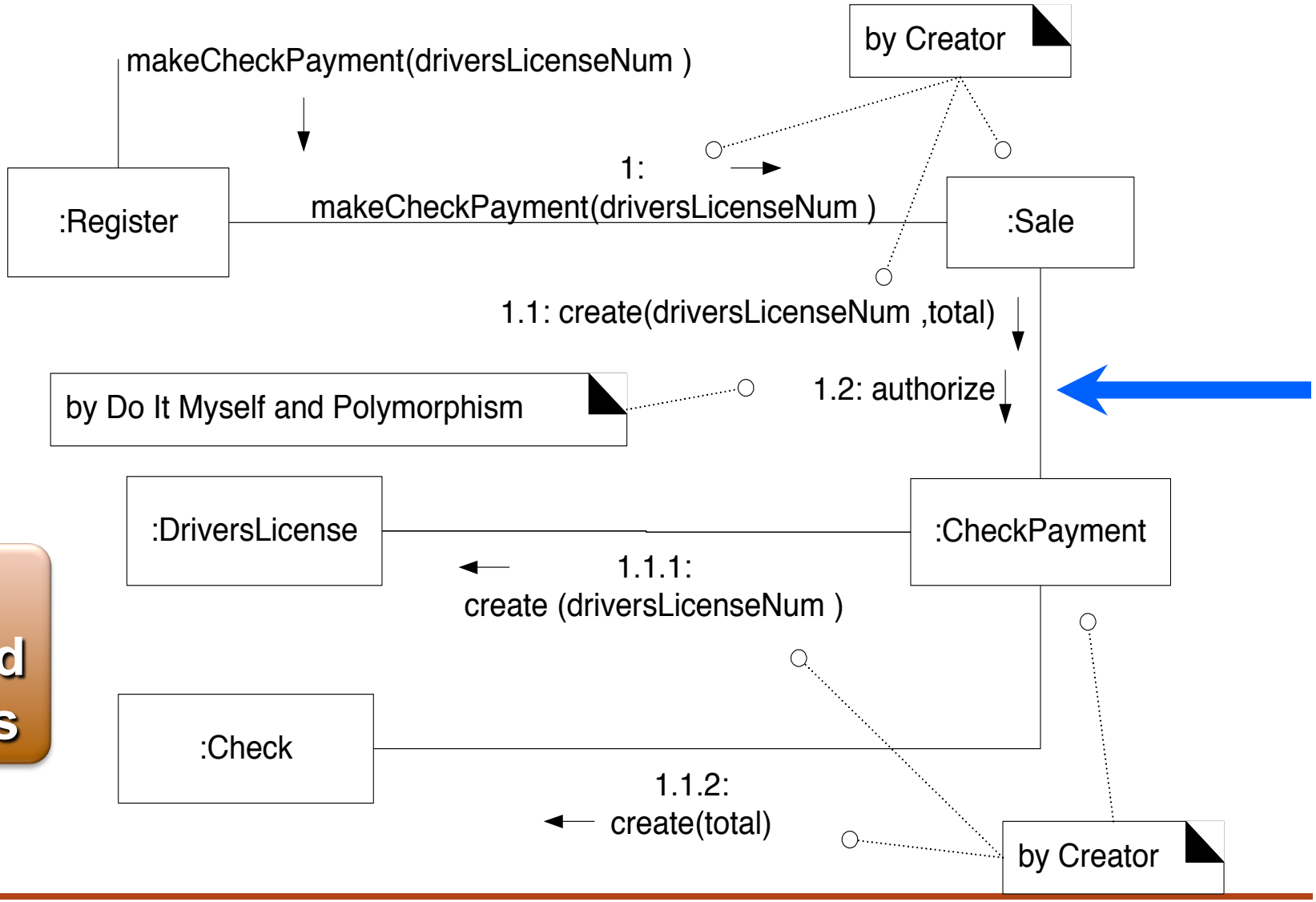
- What do we do with different payment types?
  Cash, Credit, a Check?
  - Need authorization for credit and check…
- Follow the "Do It Myself" Guideline:
  - "As a software object, I do those things that are normally done to the actual object I represent."
- A common way to apply Polymorphism and Information Expert

"Do It Myself" Example



Real world: payments are authorized
OO world: payments authorize themselves

Creating a CheckPayment



Fine-grained objects

Frameworks with Patterns

- Framework: an extendable set of objects for related functions, e.g.:
  - GUI framework
  - Java collections framework
- Provides cohesive set of interfaces & classes
  - Capture the unvarying parts
  - Provide extension points to handle variation
- Relies on the Hollywood Principle:
  - "Don't call us, we'll call you."

Designing a Persistence Framework

Domain Layer

Persistence Framework

PersistenceFaçade
get(OID, class):Object put(OID, object)

Store object in RDB

put(OID, Butler U.)

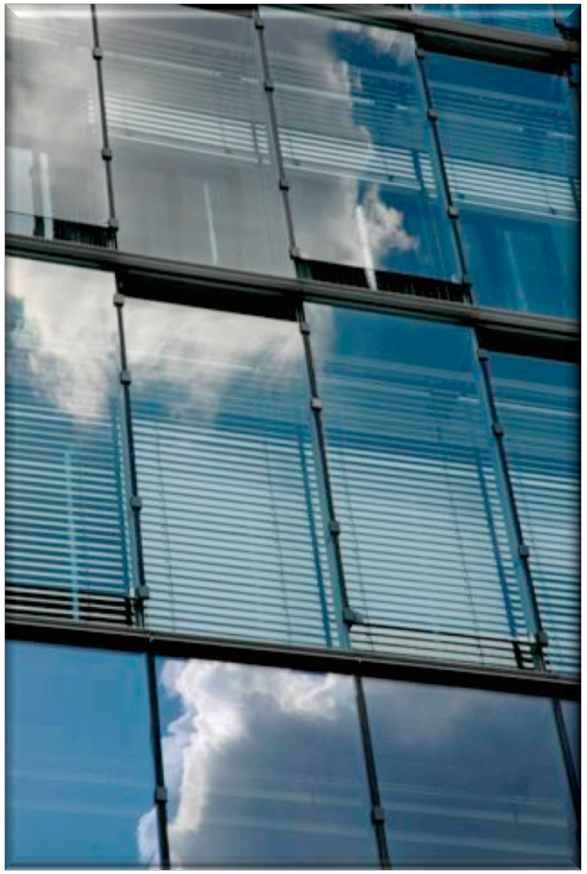
Relational Database

Name	City
RHIT	Terre Haute
Purdue	W. Lafayette
Indiana U.	Bloomington
Butler U.	Indianapolis

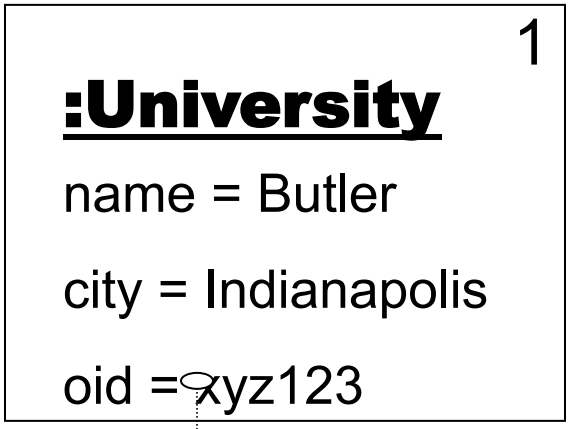
University Table

The Façade Pattern for Object ID

- Need to relate objects to database records and ensure that repeated materialization of a record does not result in duplicate objects
- Object Identifier Pattern
  - assigns an object identifier (OID) to each record
  - Assigns an OID to each object (or its proxy)
  - OID is unique to each object



Maps between Persistent Object & Database

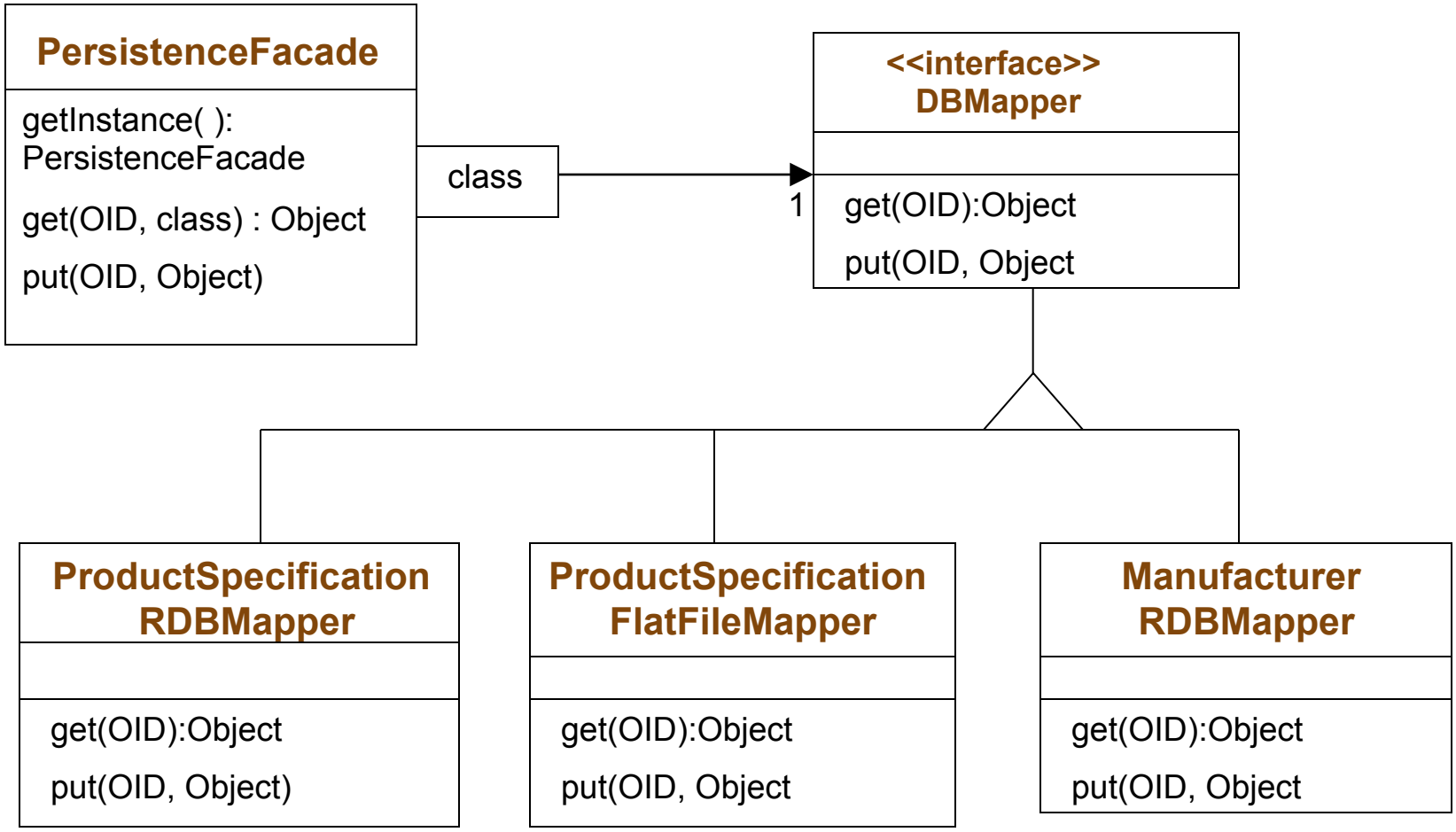


The OID may be contained in proxy object instead

University Table

OID	name	city
XI001	RHIT	Terre Haute
wxx246	Purdue	W. Lafayette
xxz357	Indiana U.	Bloomington
xyz123	Butler U.	Indianapolis

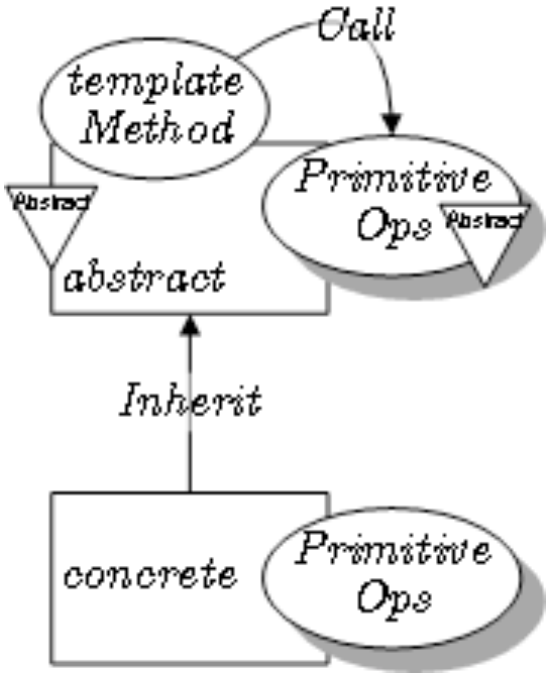
Façade Design Pattern with Brokers



Each mapper gets and puts objects in its own unique way, depending on the kind of data store and format.

Template Method Pattern

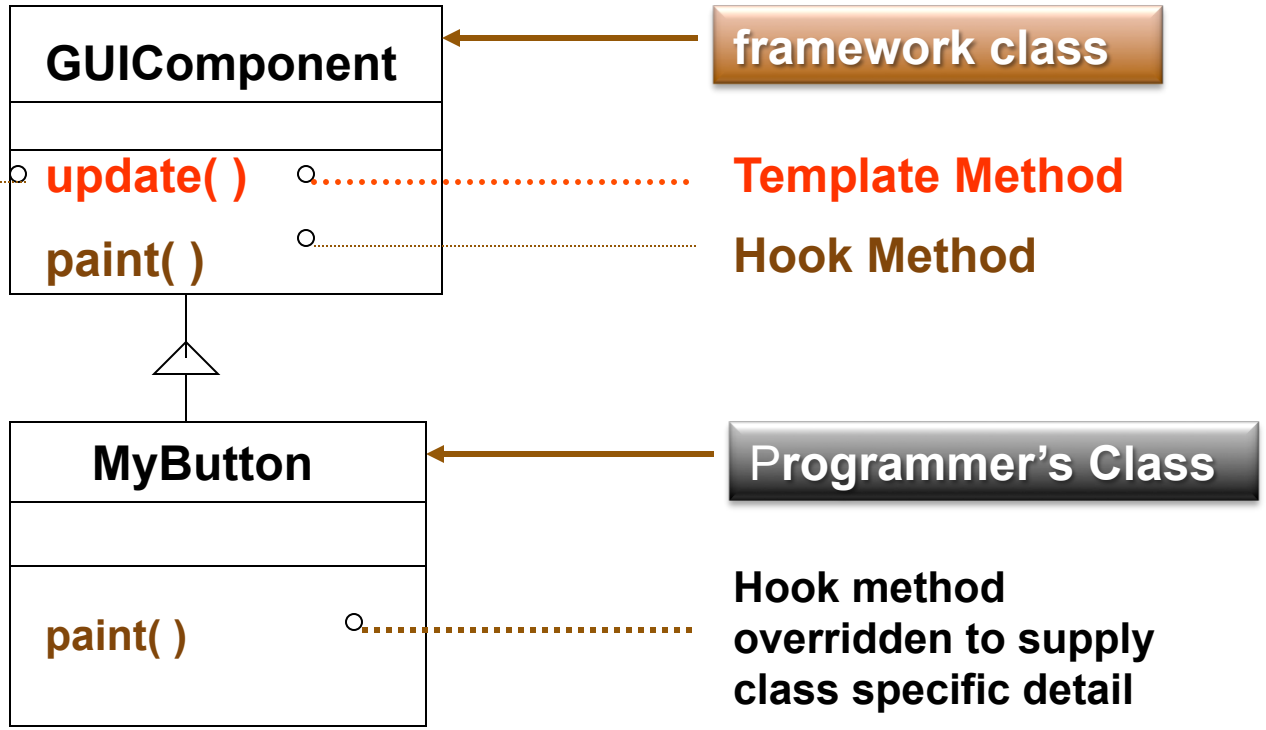
- **Problem**: How can we record the basic outline of an algorithm in a framework (or other) class, while allowing extensions to vary the specific behavior?



- **Solution**: Create a *template method* for the algorithm that calls (often abstract) helper methods for the steps. Subclasses can override/implement these helper methods to vary the behavior.

Example: Template Method used for Swing GUI Framework

```
//unvarying part of algorithm
public void update {
    clearBackground( );
    //call the hook method
    paint( );
}
```



Design Studio Calendar

	Monday	Tuesday	Thursday
8th week		Team 2.4	Team 2.1
9th week	Team 2.2	Team 2.3	Today Team 2.5
10th week	Team 2.4	Team 2.1	Course Wrap-up

Homework and Milestone Reminders

- Read Chapter 38
- Milestone 5 – Final Junior Project System and Design
  - Preliminary Design Walkthrough on Friday, February 11th, 2011 during weekly project meeting
  - Final due by 11:59pm on Friday, February 18th, 2011
- Team 2.4 Design Studio on Monday