More Object Design with GoF Patterns (continued)

Shawn Bohner
Office: Moench Room F212
Phone: (812) 877-8685
Email: bohner@rose-hulman.edu

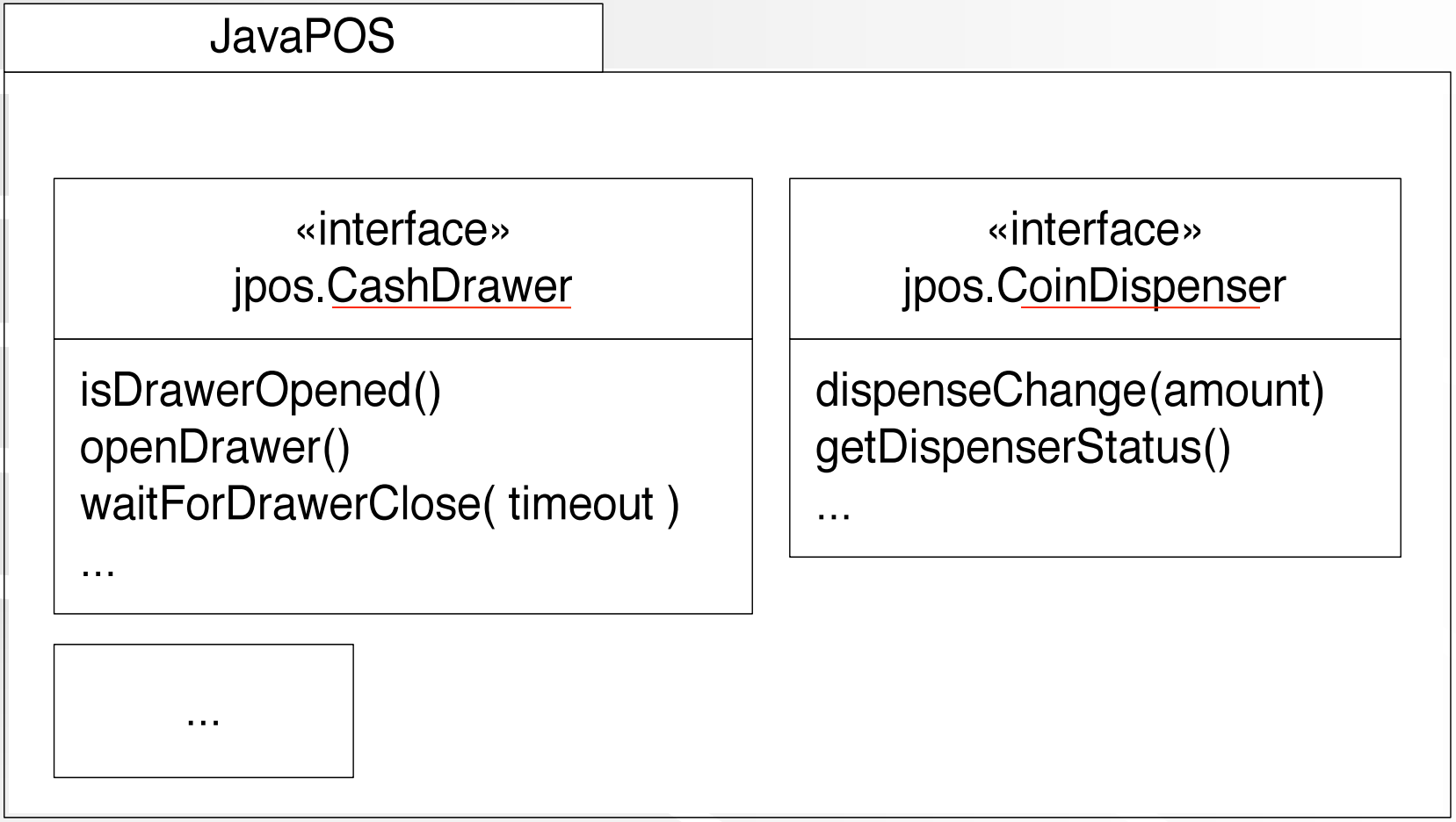
Applying Patterns to NextGen POS Iteration 3

- Local caching
  - Used Adapter and Factory
- Failover to local services
  - Used Proxy, Adapter, and Factory
- **Support for third-party POS devices**
- **Handling payments**

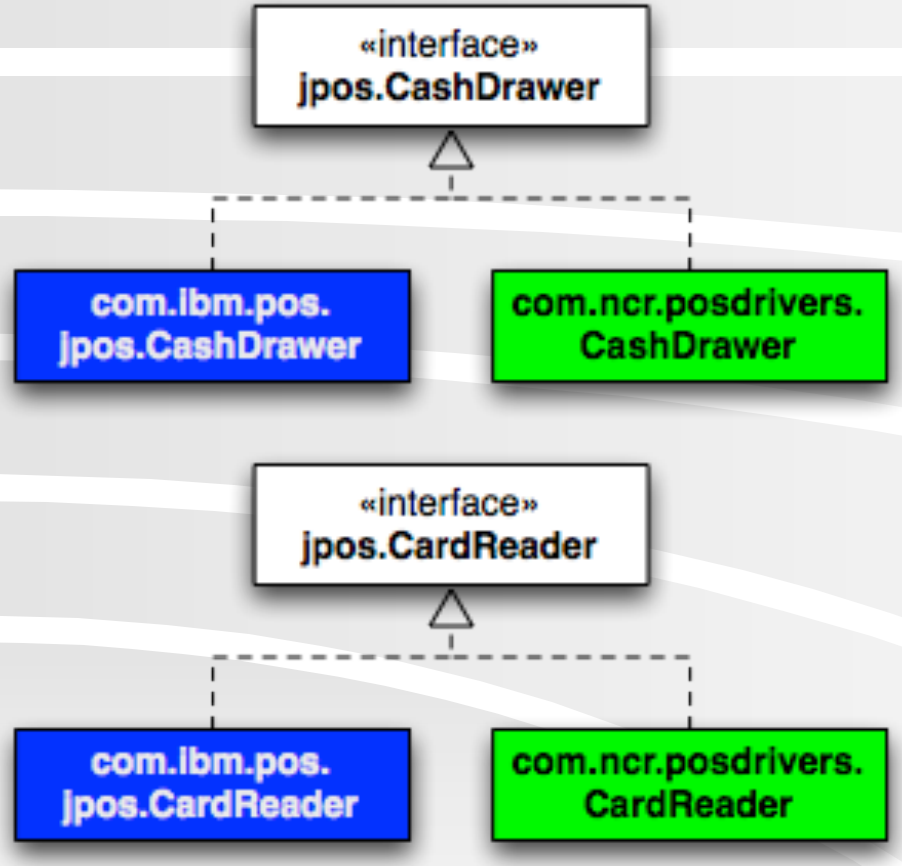
Accessing External Physical Devices

- 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?

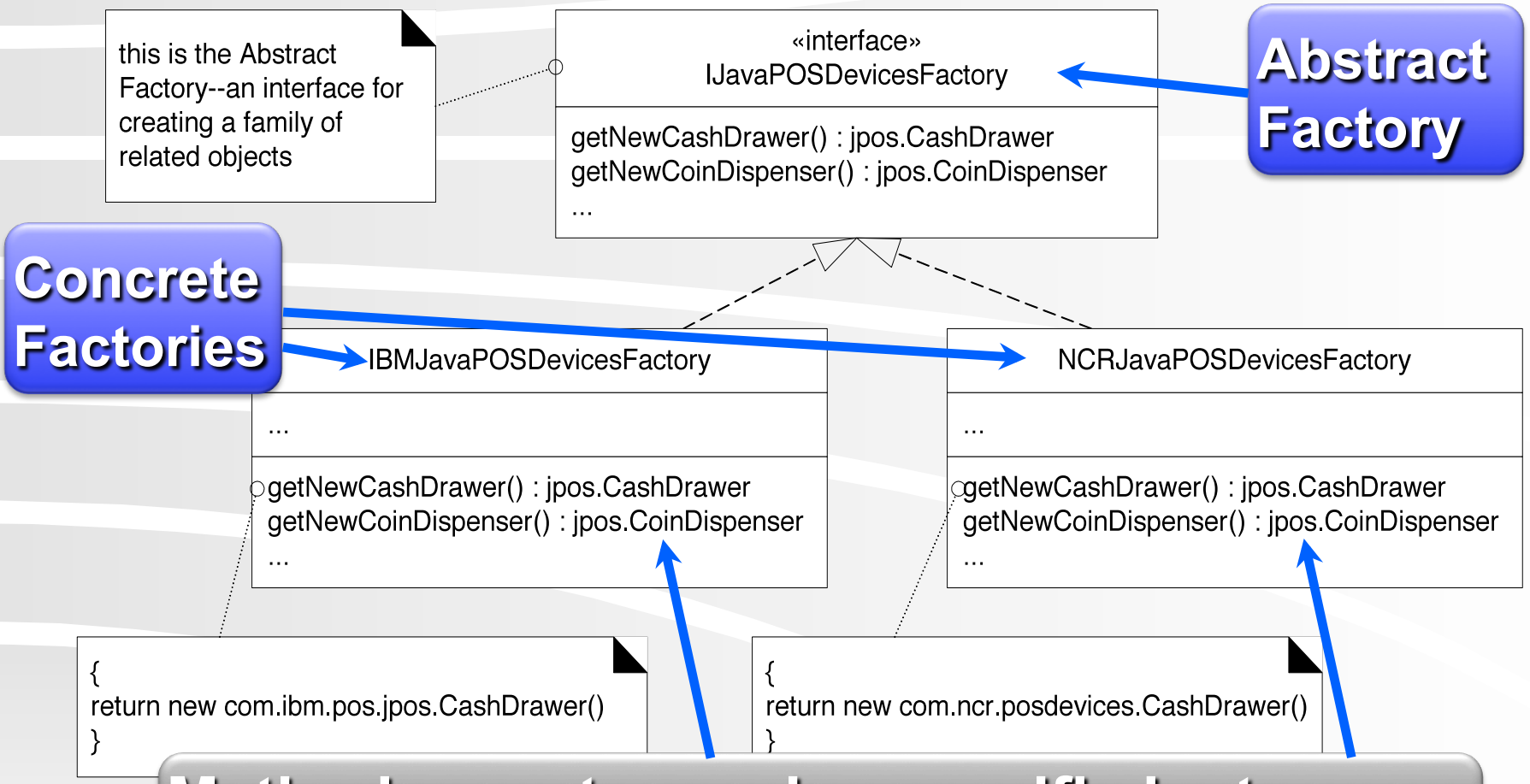
Assume: A given store uses a single manufacturer

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



ROSE-HULMAN INSTITUTE OF TECHNOLOGY

First Attempt at Using Abstract Factory

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

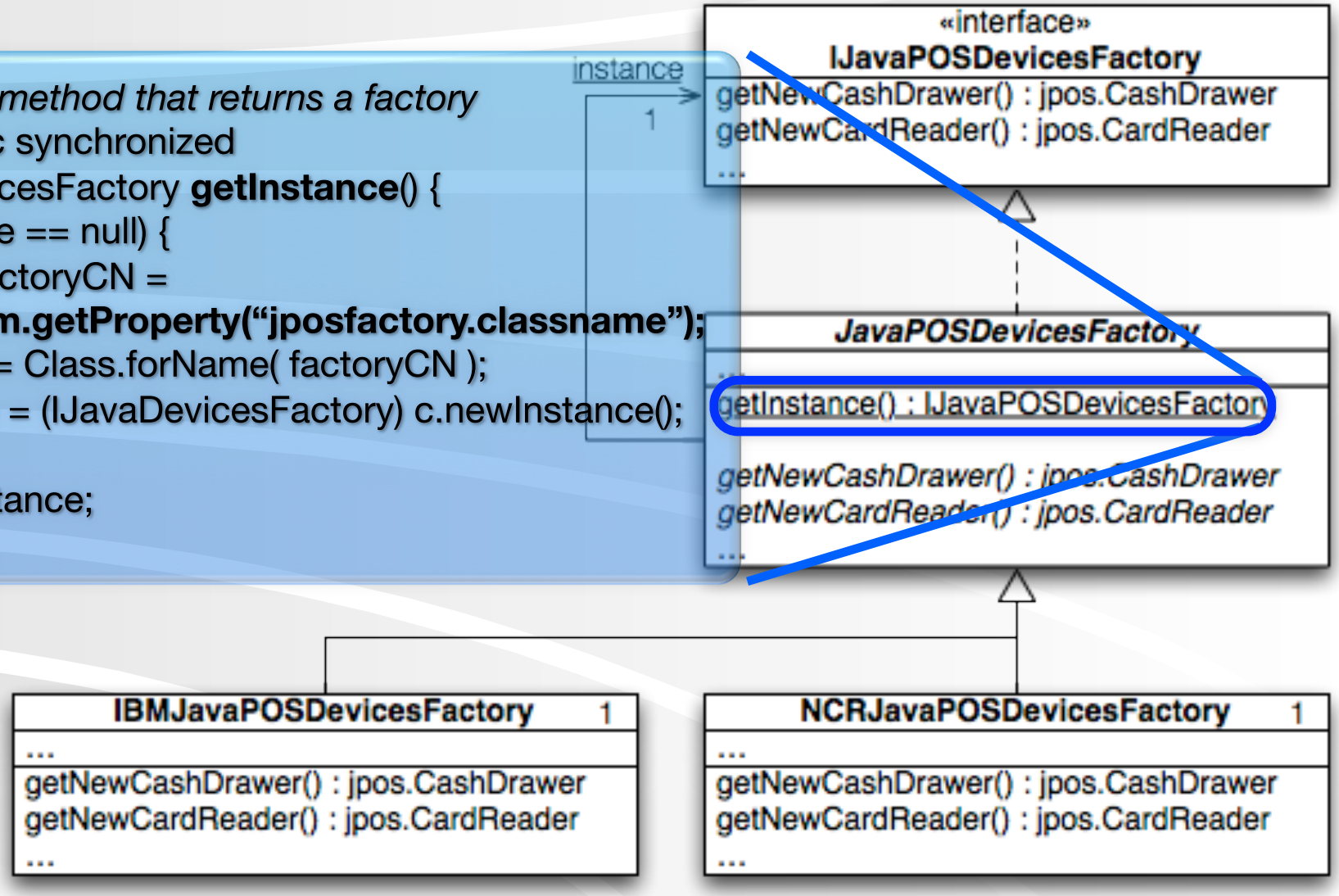
Constructs a vendor-specific concrete factory

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;
}
```



ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Using a Factory Factory

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

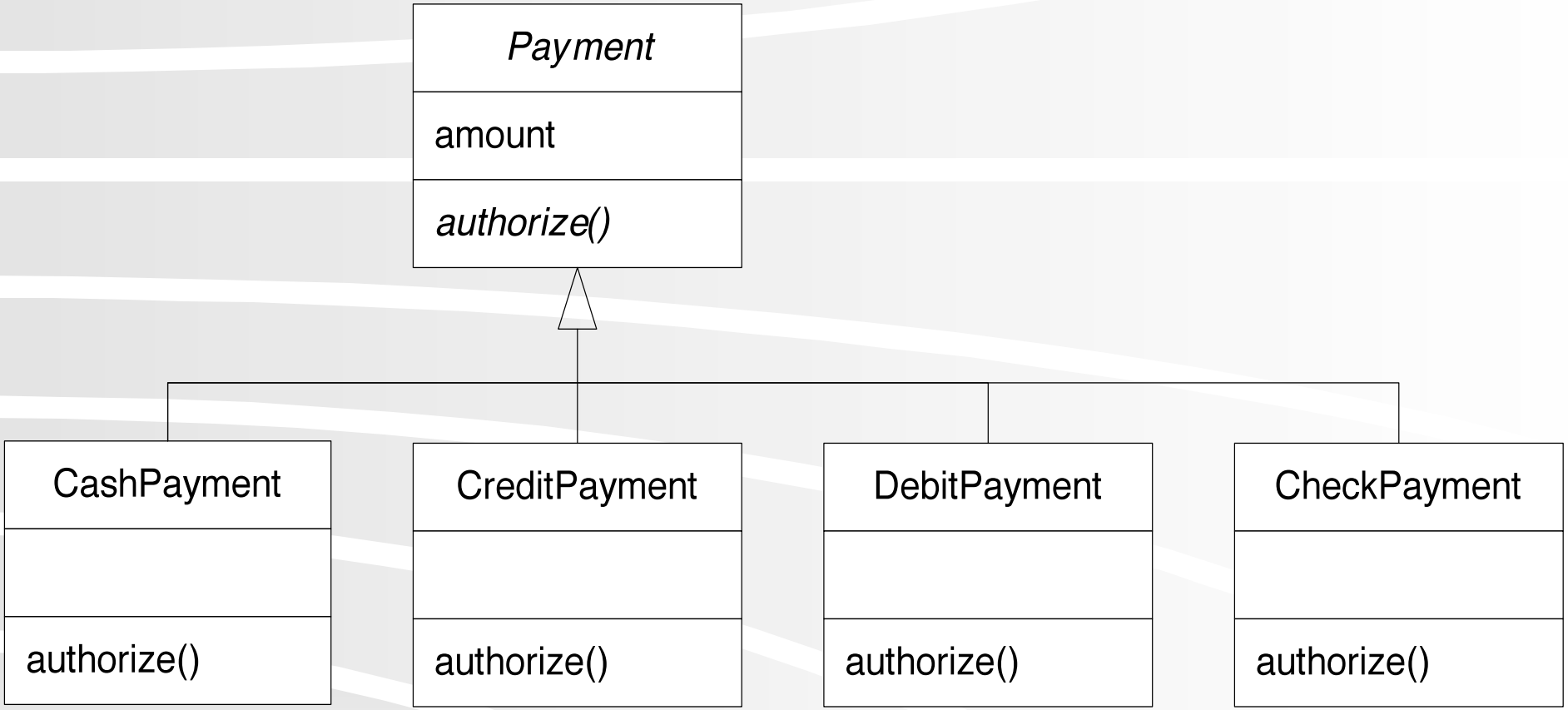
Gets a vendor-specific concrete factory singleton

Uses it to construct device instances

Handling Payments

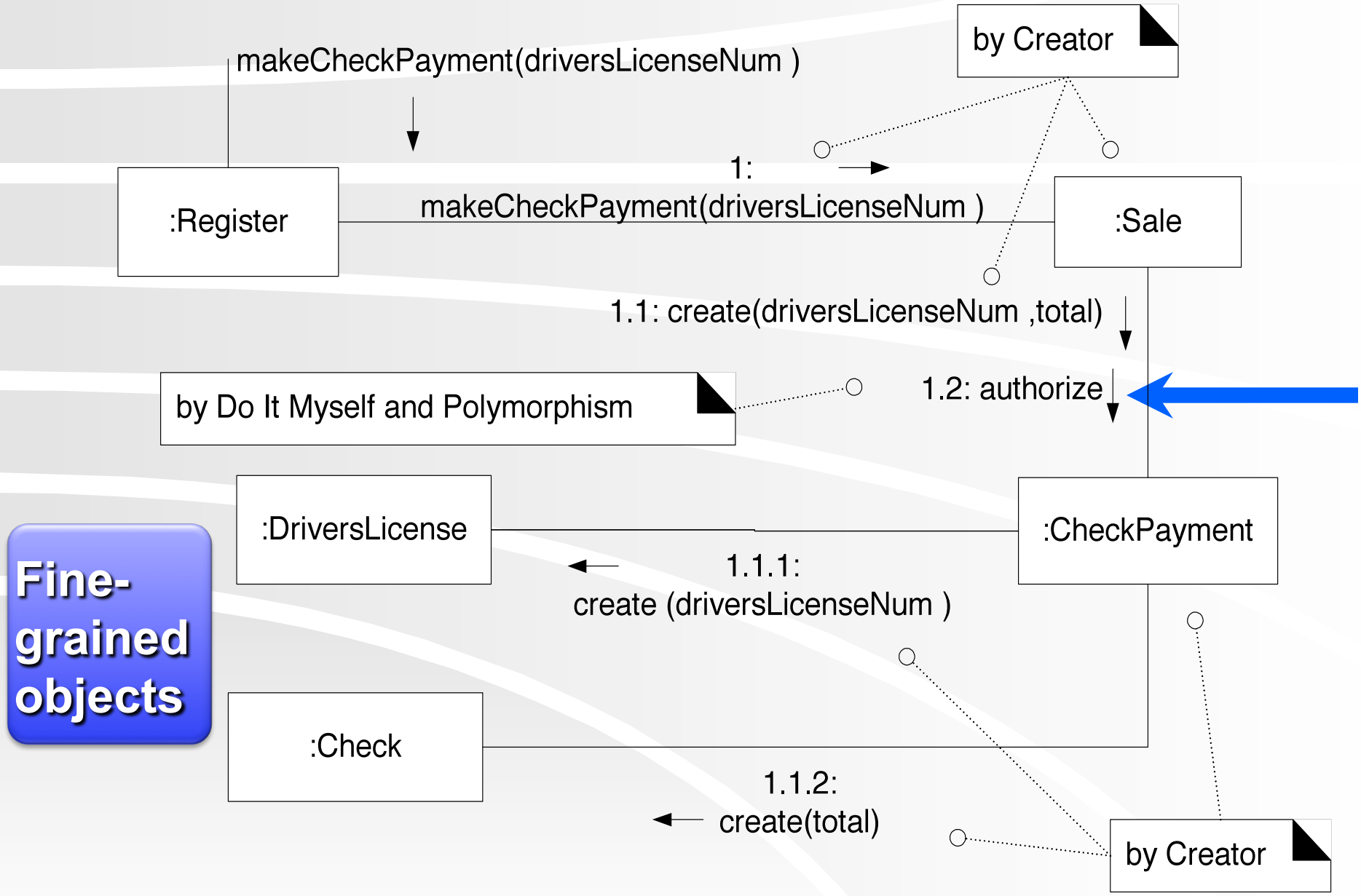
- What do we do with different payment types? Cash, Credit, a Check?
  - Need authorization for credit and check…
- Follow the "Do It Myself" Guideline:
  - "As a software object, I do those things that are normally done to the actual object I represent."
- A common way to apply Polymorphism and Information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

“Do It Myself” Example

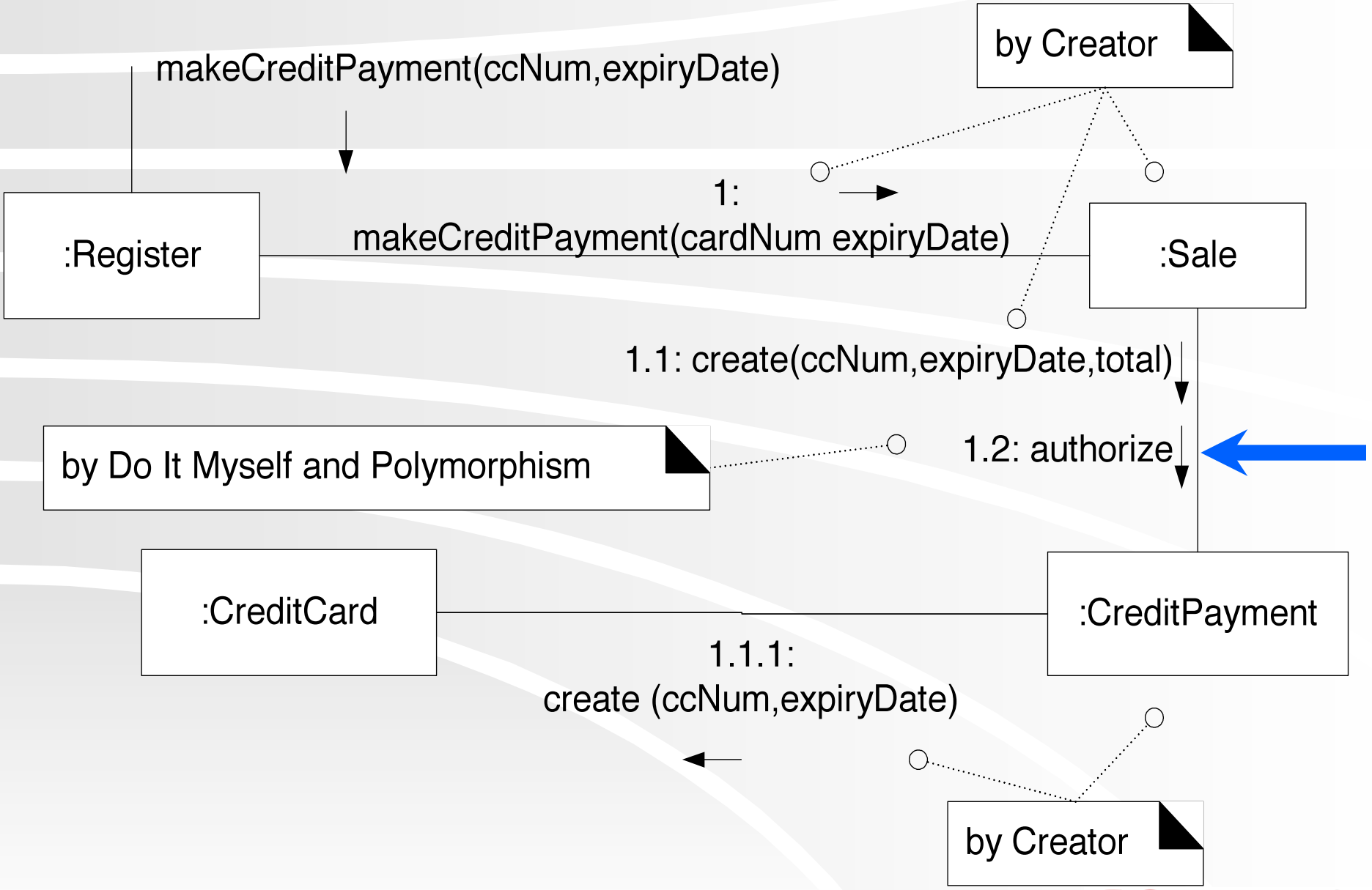


Real world: payments are authorized
OO world: payments authorize themselves

Creating a CheckPayment



Creating a CreditPayment



Frameworks with Patterns

- Framework: an extendable set of objects for related functions, e.g.:
  - Swing 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

Accessing Persistence Service via Façade

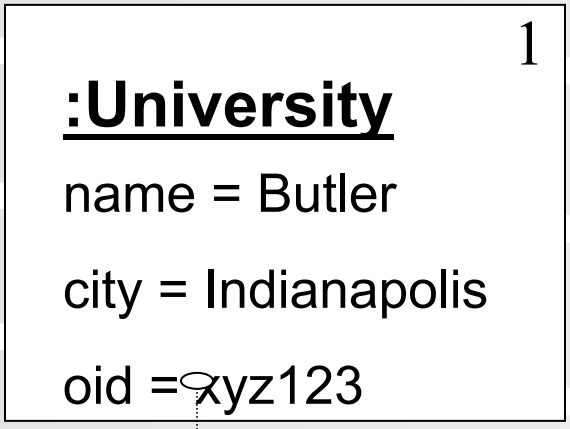
- Unified interface to set of interfaces in a subsystem
- Façade defines a higher-level interface that makes the subsystem easier to use
- Façade Applications:
  - Layer the subsystem using Facade to define an entry point to each subsystem level
  - Introduce a Facade to decouple subsystems from clients and other subsystems
    - Promotes independence and portability
  - Façade produces simple default view of subsystem

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

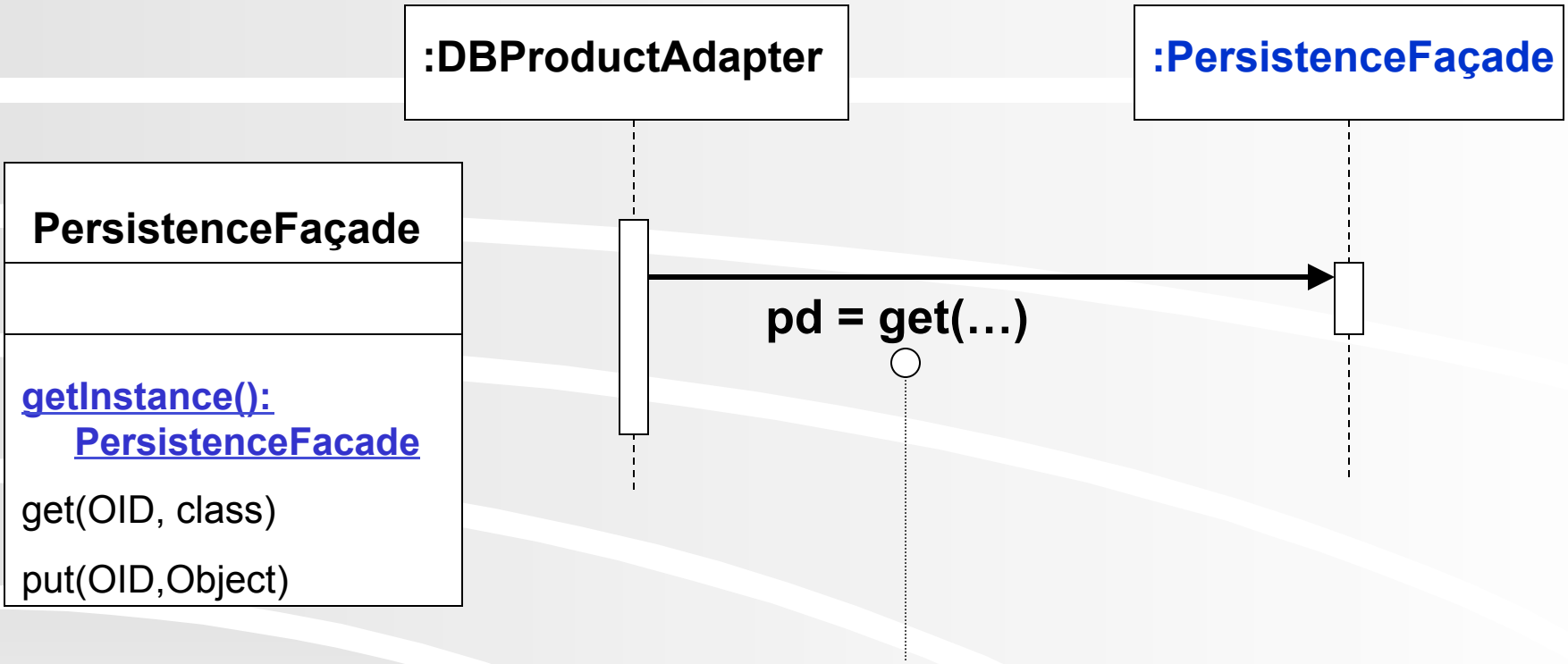


The OID may be contained in proxy object instead

University Table

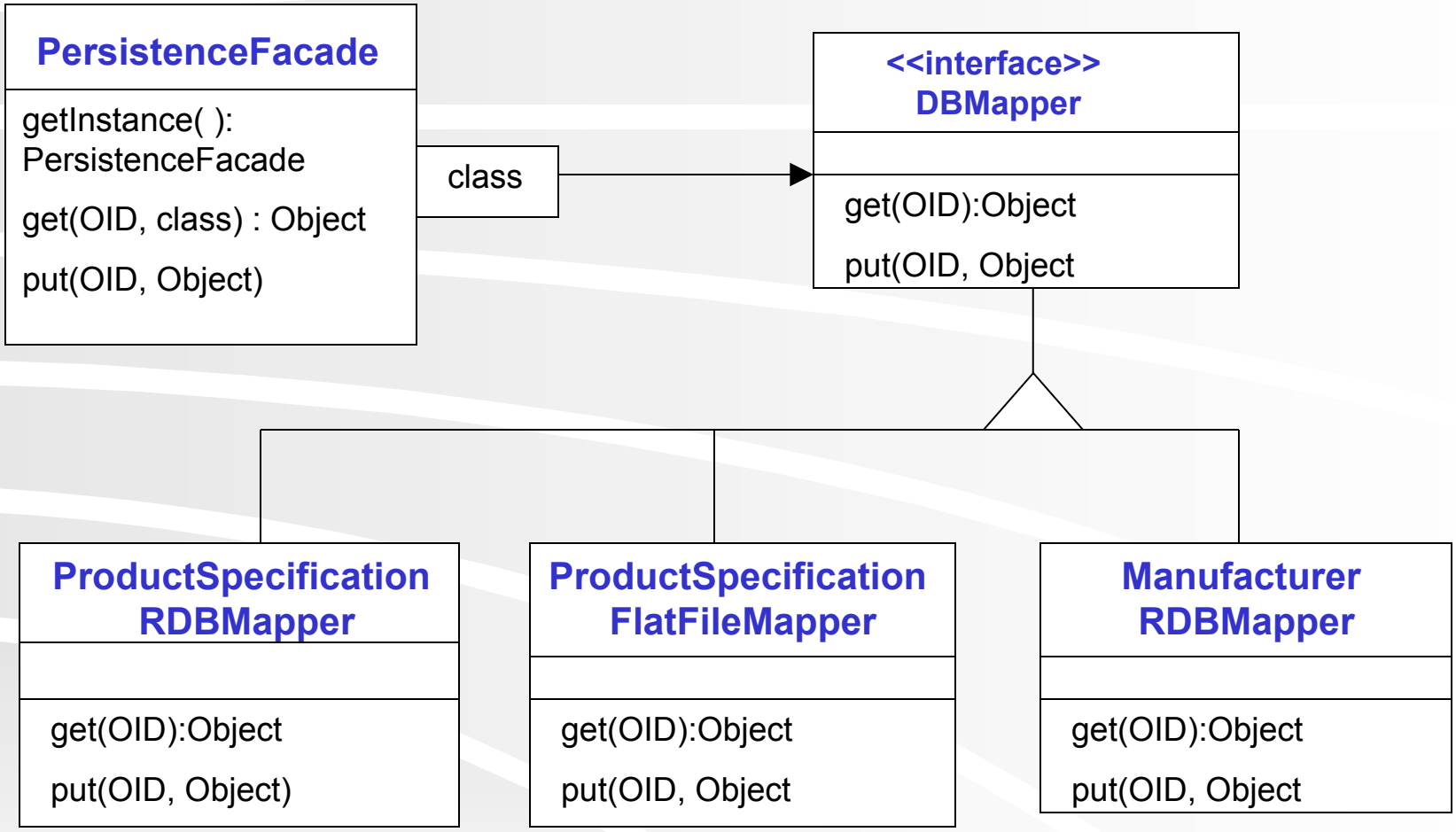
OID	name	city
XI001	RHIT	Terre Haute
wxx246	Purdue	W. Lafayette
xxz357	Indiana U.	Bloomington
xyz123	Butler U.	Indianapolis

The Persistence Façade




```
//example use of the facade
OID = new OID("XYZ123");
ProductDescription pd = (ProductDescription)
PersistenceFacade.getInsance().get(oid, ProductDescription.class);
```

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.

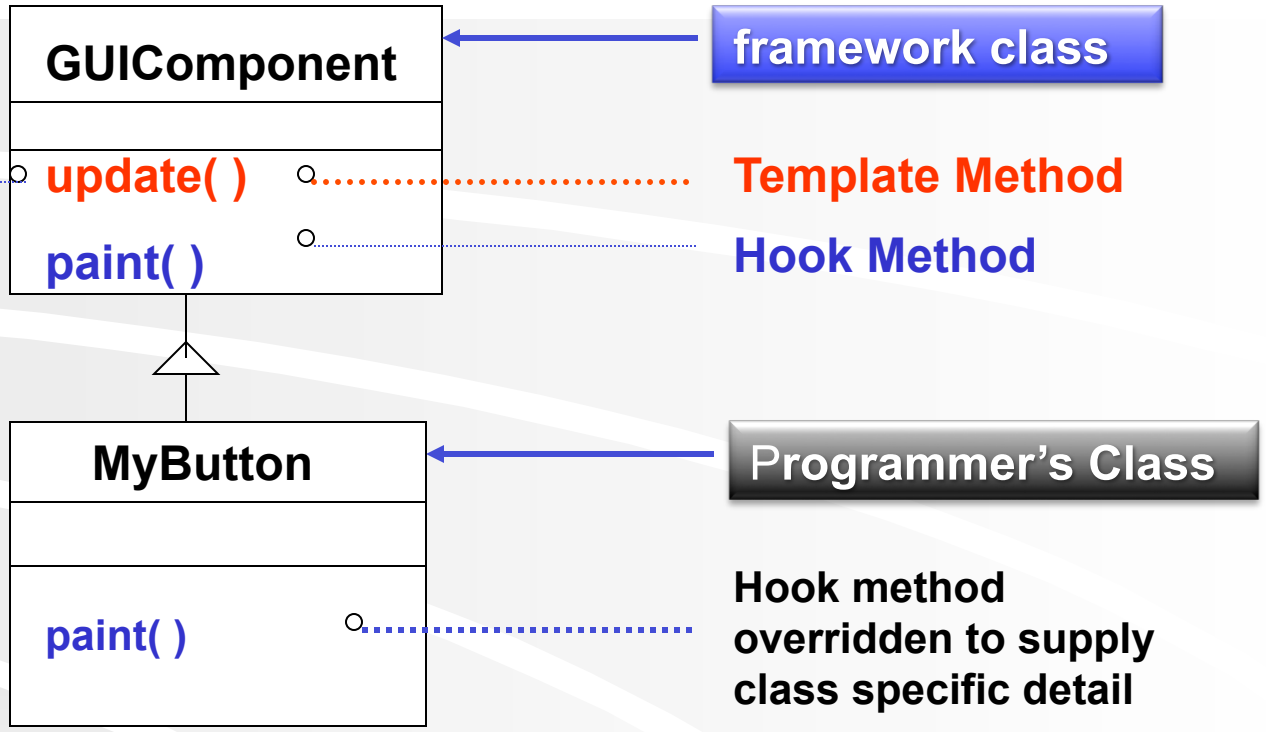
Template Method Pattern

- **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:
Team 13: CSSE Portfolio

~5 minutes:
Team describes problem and current solution (if any)

~3 minutes:
Class thinks about questions, alternative approaches Q7

~12 minutes:
On-board design with team modeling and instructor advising/facilitating

Homework and Milestone Reminders

- Read Chapter 38
- Milestone 5 – Iteration 3 Junior Project System with finalized Design Document
  - Preliminary Design Walkthrough on Friday, February 12th, 2010 during project meeting.
  - Final Project Due by 11:59pm Friday, February 19th, 2010.