Persistence Frameworks with GoF Patterns (State & Command)

Shawn Bohner

Office: Moench Room F212

Phone: (812) 877-8685

Email: bohner@rose-hulman.edu

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Final Exam - Optional

Email me by Tuesday, Feb. 16th, to sign up for Final Exam.

- Monday, Feb. 22nd, at 8am
- If you don't take the exam, we'll use your exam 1 grade as your final exam grade
- If you sign-up, you must take the exam
- Taking the exam can improve or lower your grade

Plan for Today

- Short survey on projects
- Finish up Template Pattern
- State Pattern
- Command Pattern
- Design Studio—Team 15: Code Assistant

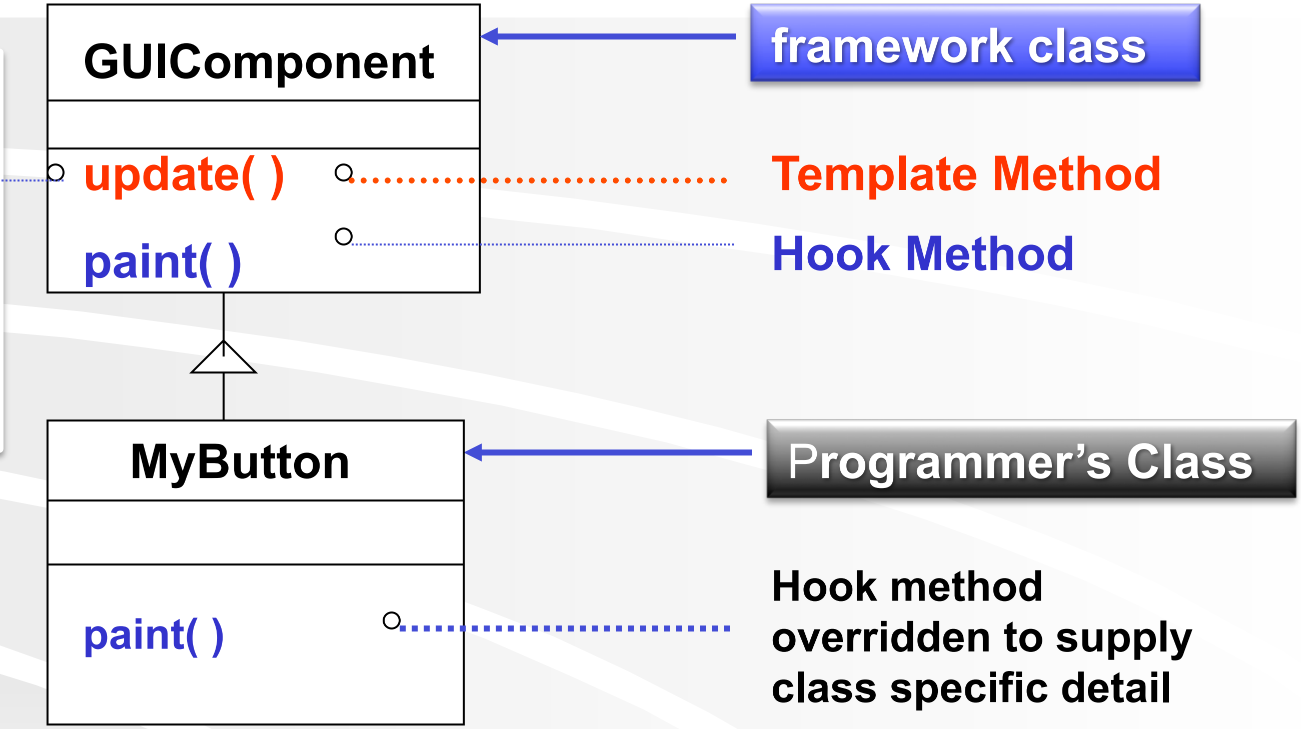
Template Method Pattern

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( );
}
```



framework class

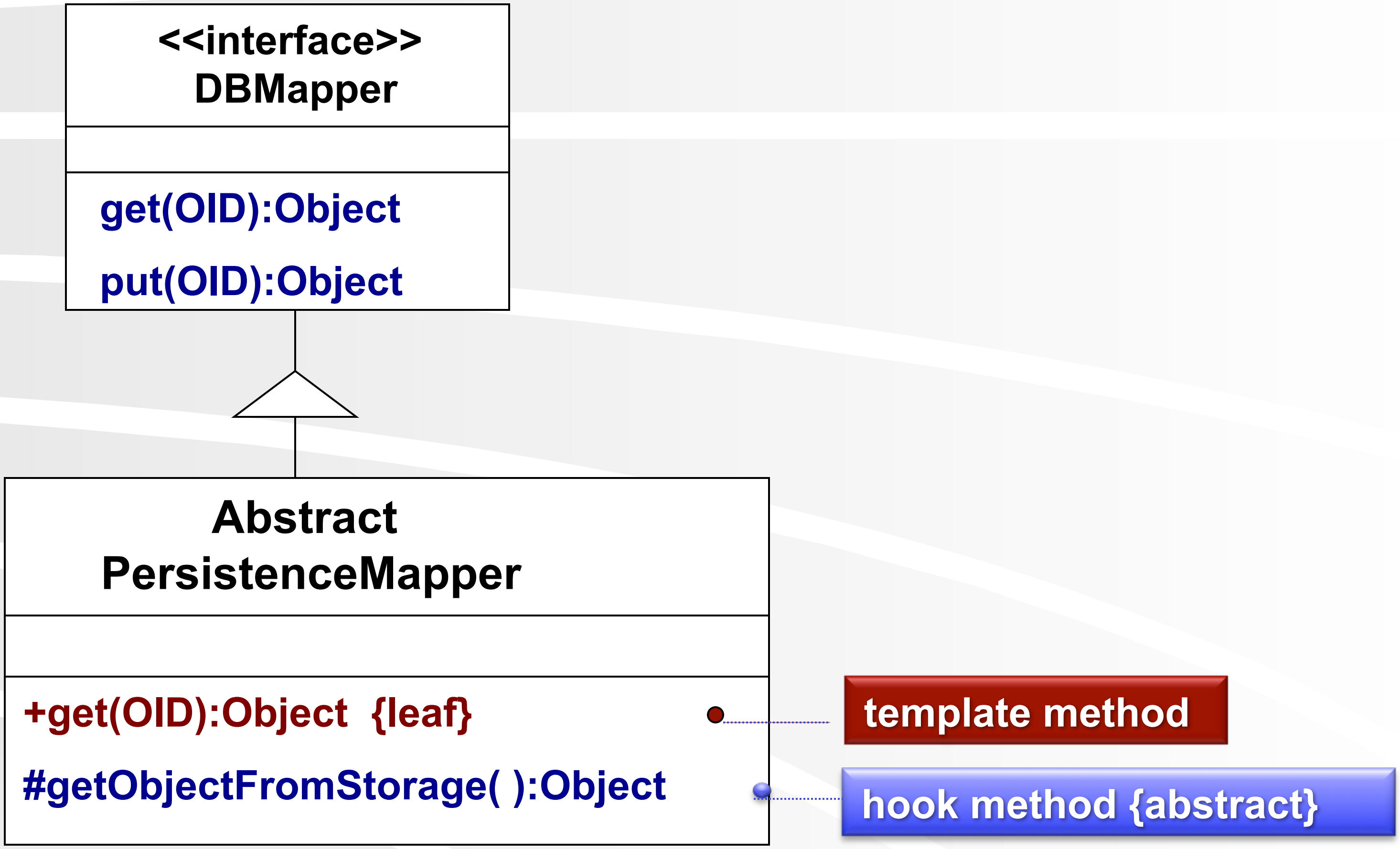
Template Method

Hook Method

Programmer’s Class

Hook method
overridden to supply
class specific detail

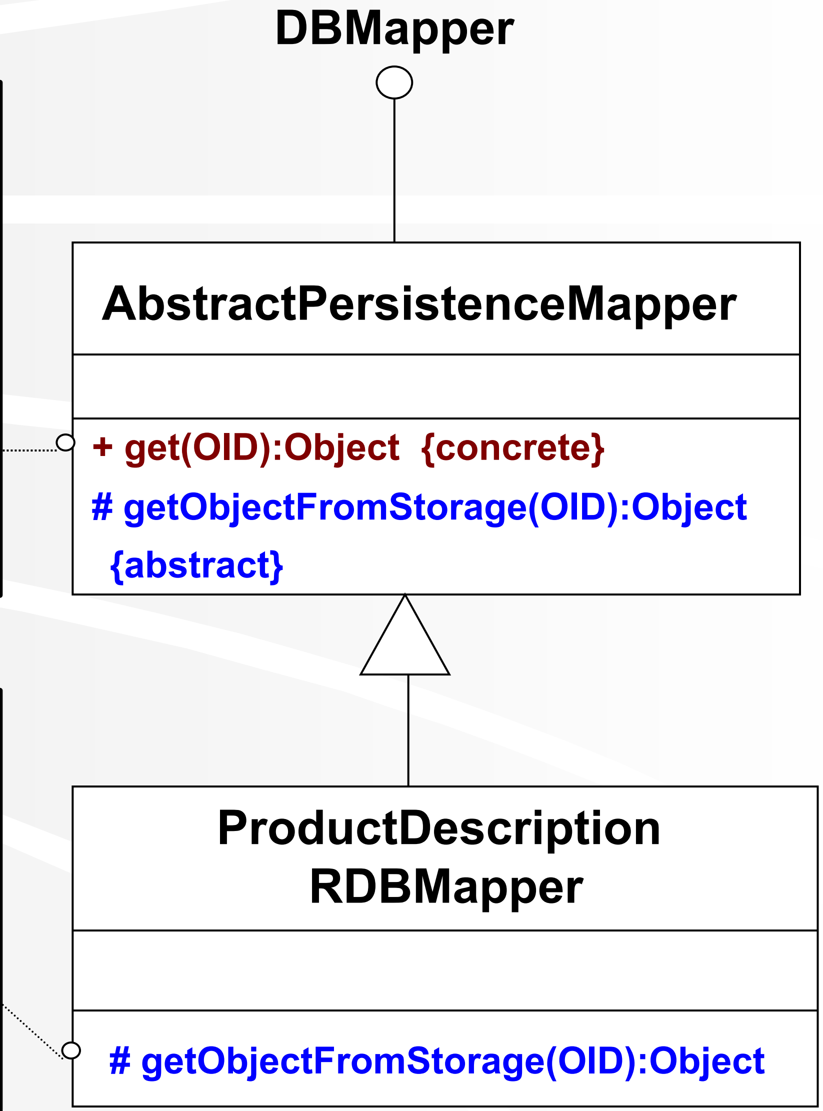
Template Method in NexGen POS (1 of 2)



Template Method in NexGen POS (2 of 2)

```
//template method
public final Object get(OID oid) {
     obj = cachedObjects.get(oid);
     if (obj == null) {
          //hook method
          obj = getObjectFromStorage(oid);
          cachedObject.put(oid, obj);  }
      return obj;  }
```

```
//hook method override
protected Object getObjectFromStorage(OID oid) {
     String key = oid.toString( );
     dbRec = SQL execution result of
          “Select* from PROD_DESC where key =“
          +key
ProductDescription = new ProductDescription();
pd.setPrice(dbRec.getColumn(“PRICE”); etc
```



DBMapper

AbstractPersistenceMapper

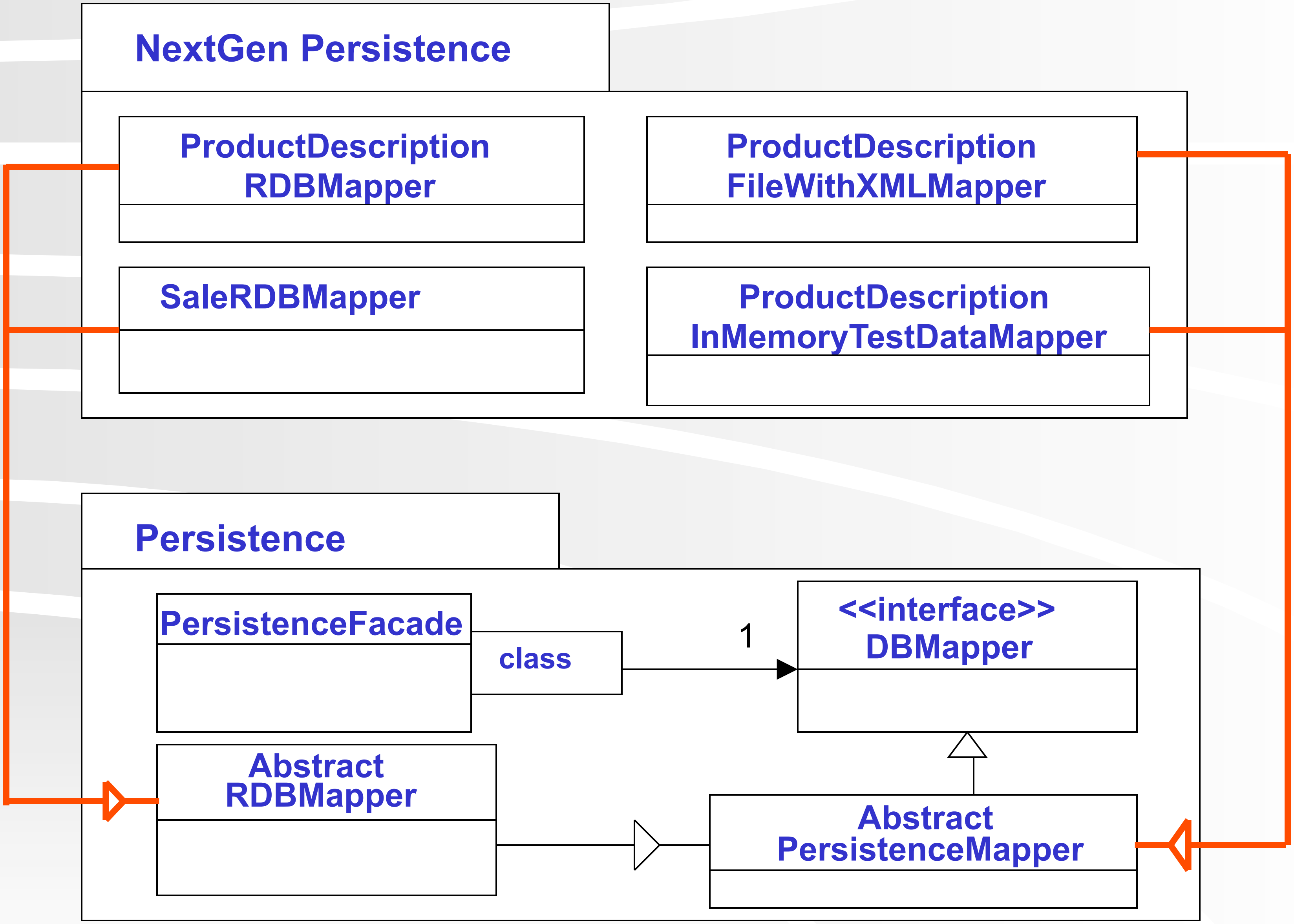
+ get(OID):Object {concrete}

# getObjectFromStorage(OID):Object {abstract}

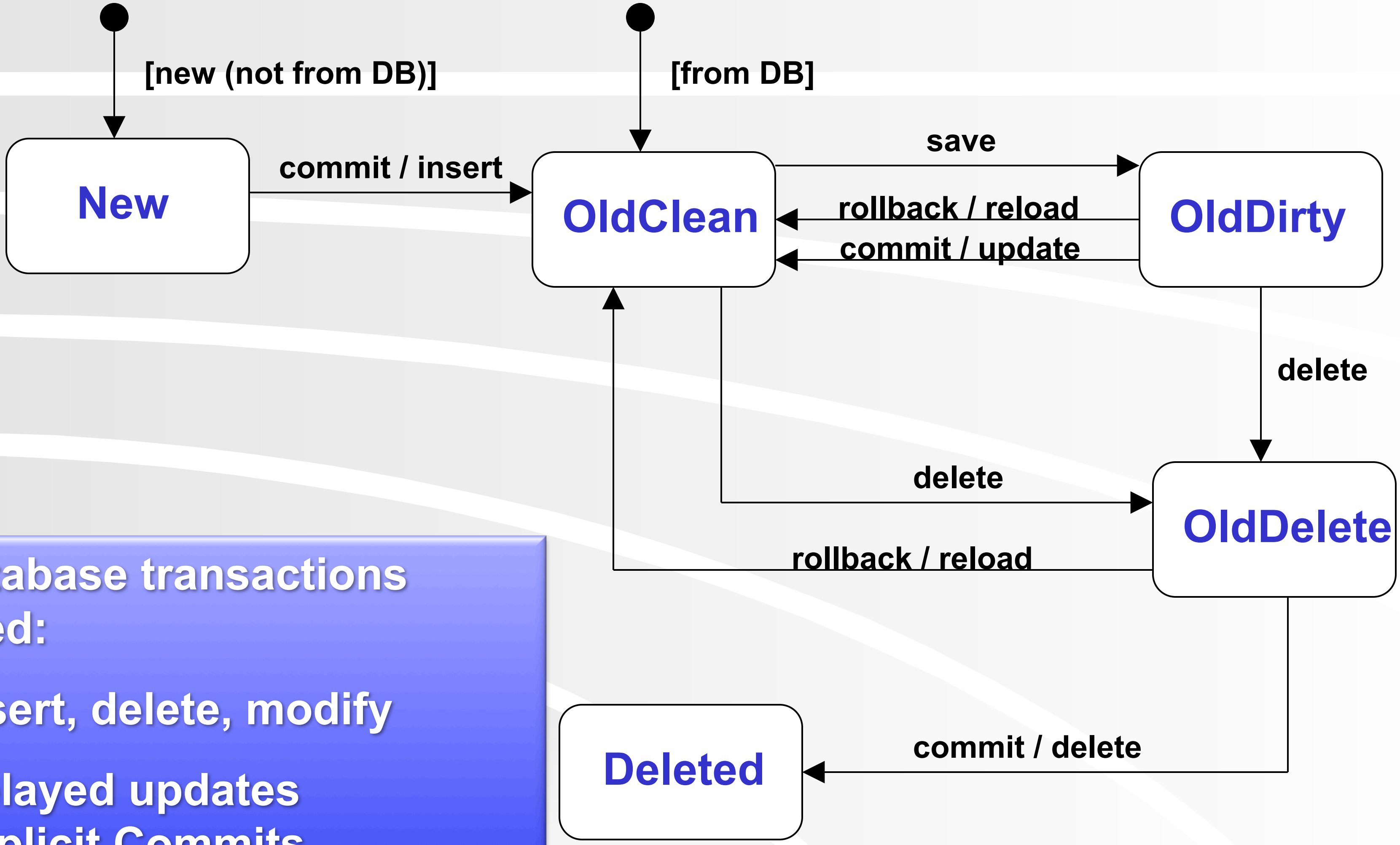
ProductDescription RDBMapper

# getObjectFromStorage(OID):Object

Persistence Framework


NextGen Persistence
ProductDescription RDBMapper
ProductDescription FileWithXMLMapper
SaleRDBMapper
ProductDescription InMemoryTestDataMapper
Persistence
PersistenceFacade
class
1
<<interface>> DBMapper
Abstract RDBMapper
Abstract PersistenceMapper

Transactional States & the State Pattern



Database transactions need:

-insert, delete, modify

-Delayed updates /Explicit Commits (rollback)

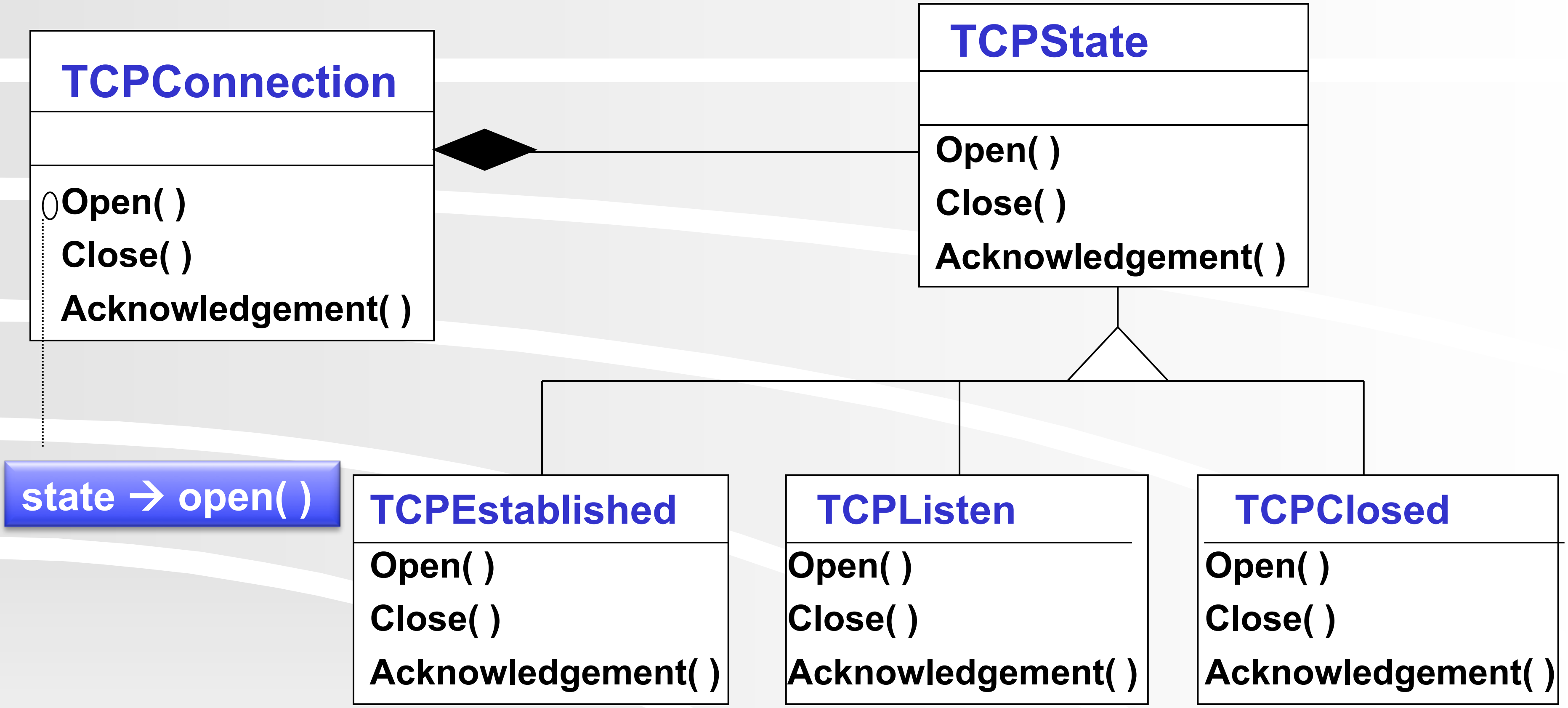
State Pattern

Problem: When the behavior of an object, obj, changes depending on its state, how can we avoid complicated conditional statements?

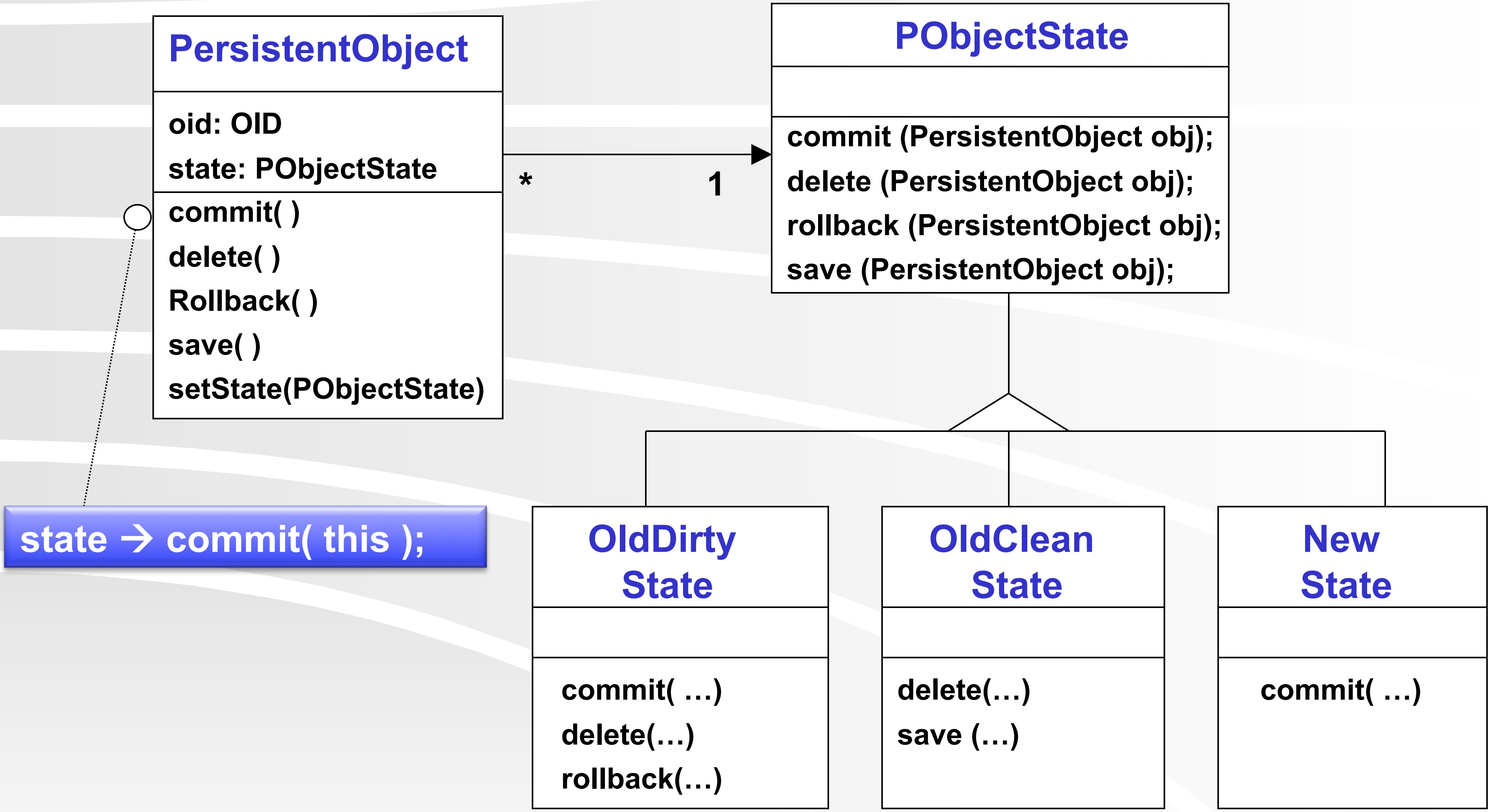
Solution: Create *state classes* implementing a common interface. Delegate state-dependent methods from obj to the current state object.

Q1,2

Example: State Pattern in TCP



State Patten in Persistence Framework



PersistentObject

oid: OID

state: PObjectState

commit( )

delete( )

Rollback( )

save( )

setState(PObjectState)

*

1

PObjectState

commit (PersistentObject obj);

delete (PersistentObject obj);

rollback (PersistentObject obj);

save (PersistentObject obj);

state → commit(this);

OldDirty State

commit(…)

delete(…)

rollback(…)

OldClean State

delete(…)

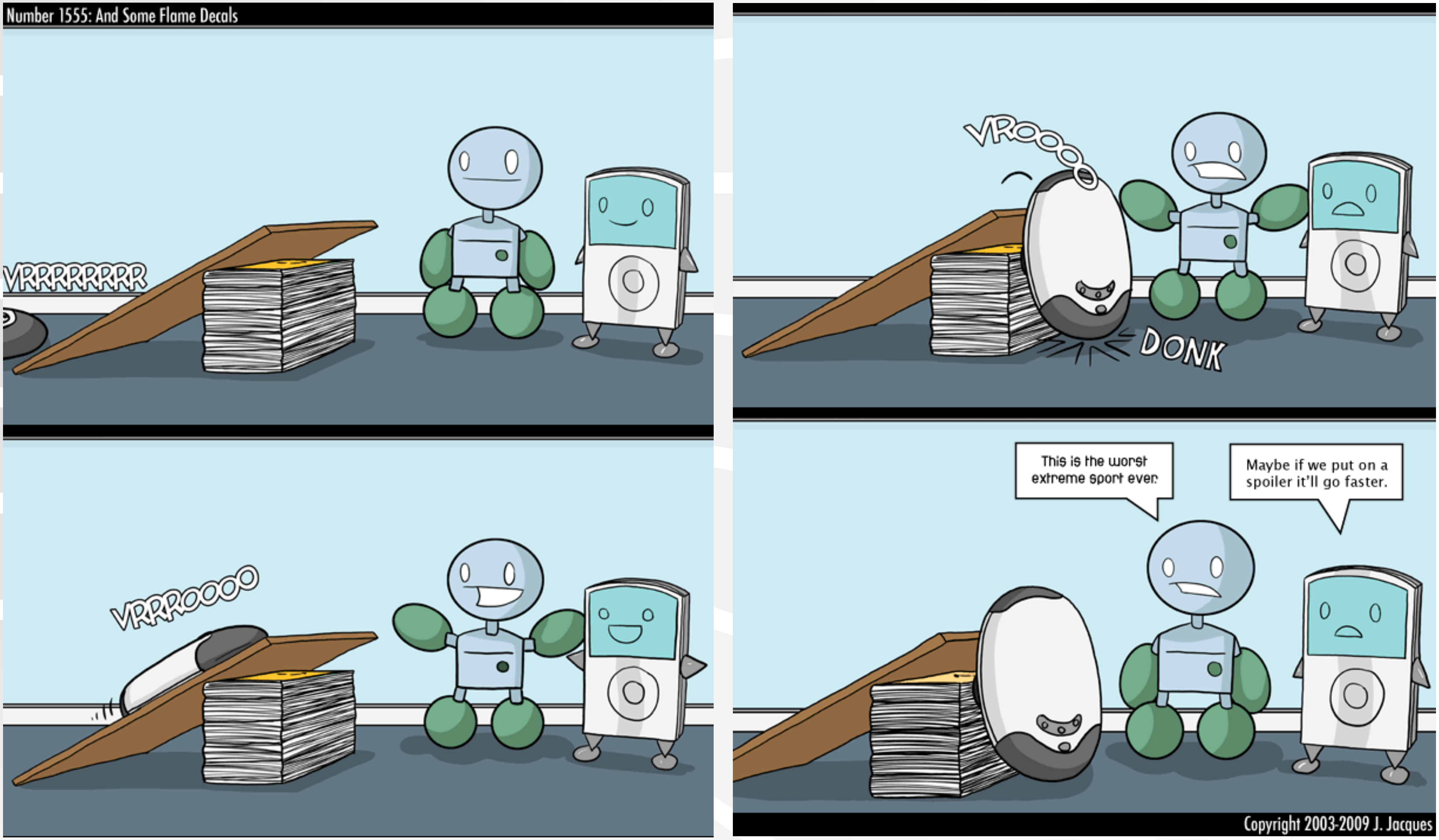
save (…)

New State

commit(…)

Q3

Cartoon of the Day


Number 1555: And Some Flame Decals
VRRRRRRRR
VRRROOOO
VROOOO
DONK
This is the worst extreme sport ever.
Maybe if we put on a spoiler it'll go faster.
Copyright 2003-2009 J. Jacques

Used by permission. http://www.questionablecontent.net/view.php?comic=1555

Command Pattern

Problem: When we need to record operations so we can undo them, or execute them later, what should we do?

Solution: Define a Command interface that represents all possible operations. Create subclasses of it for each kind of operation and instances for each actual operation.

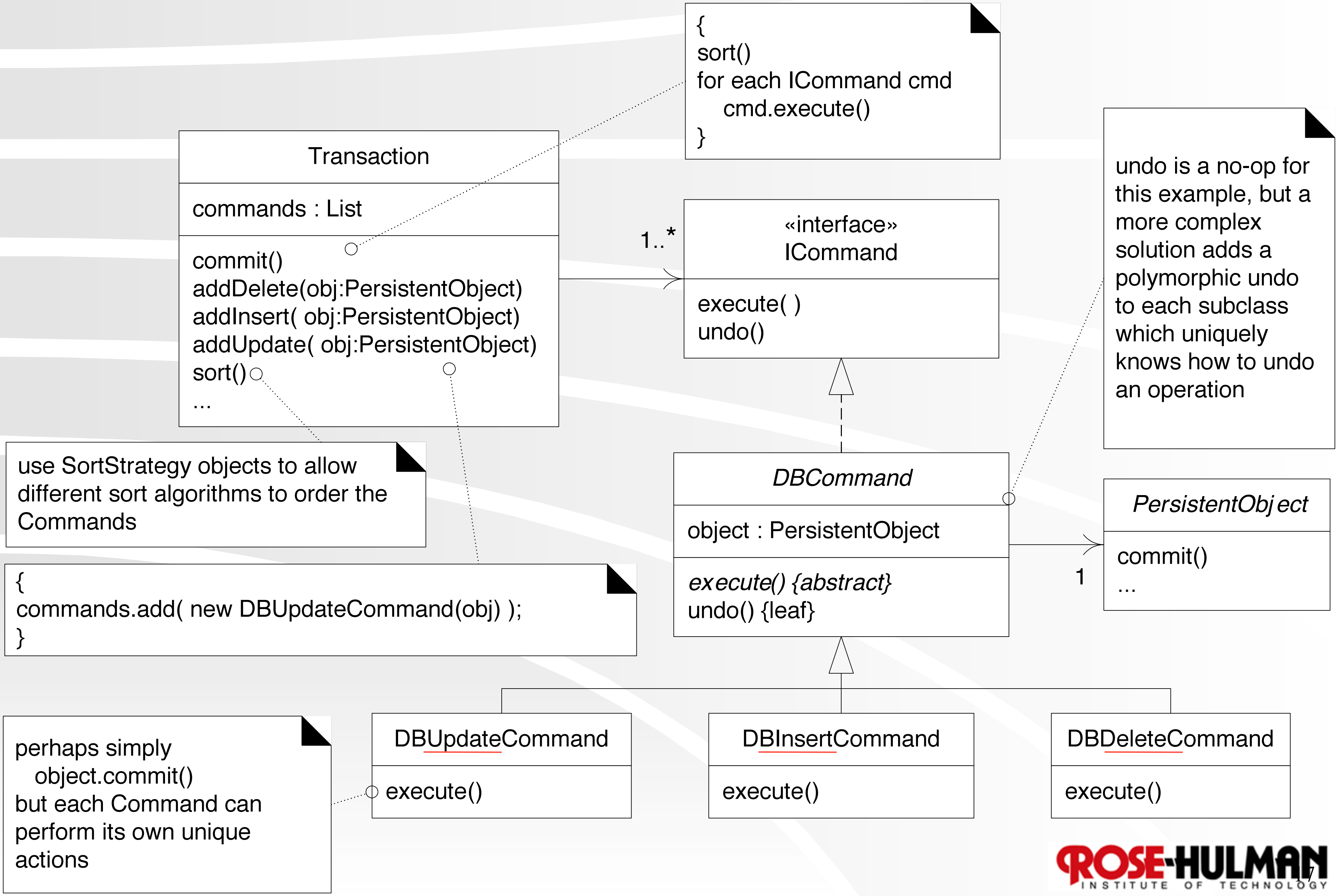
Q4,5

Uses for the Command Pattern

- Undo/redo
- Prioritizing and Queuing operations
- Composing multi-part operations
- Progress bars
- Macro recording

Q6

Command Pattern in NextGen POS



```
{
sort()
for each ICommand cmd
   cmd.execute()
}
```

Transaction

commands : List

commit()
addDelete(obj:PersistentObject)
addInsert(obj:PersistentObject)
addUpdate(obj:PersistentObject)
sort()
...

1..*

«interface»
ICommand

execute( )
undo()

undo is a no-op for this example, but a more complex solution adds a polymorphic undo to each subclass which uniquely knows how to undo an operation

use SortStrategy objects to allow different sort algorithms to order the Commands

DBCommand

object : PersistentObject

execute() {abstract}
undo() {leaf}

PersistentObject

commit()
...

1

```
{
commands.add( new DBUpdateCommand(obj) );
}
```

perhaps simply
   object.commit()
but each Command can perform its own unique actions

DBUpdateCommand

execute()

DBInsertCommand

execute()

DBDeleteCommand

execute()

Design Studio:
Team 15: Code Assistant

~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
  - Final Project Due by 11:59pm Friday, February 19th, 2010.