CSSE 374: Persistent Frameworks with GoF Design Patterns & Deployment Diagrams

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

- 8:00am on Wednesday, Feb. 23rd
  - Room G317

- If you don’t take the exam, we’ll use your first exam grade as your final exam grade

- Sign-up for exam by Tuesday of 10th week
  - If you sign-up, you must take the exam
  - Taking the exam can improve or lower your grade

Email me by tomorrow, Feb. 15th, to sign up for Final Exam.
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
  - Finish up Template Pattern
  - State Pattern
  - Command Pattern

- Deployment Diagrams

- Design Studio with Team 2.5
Persistence Framework – a service to provide object to record mapping

In a Persistence Framework a record is to an object, as a __________ is to a graphical object in a GUI Framework.

- Think for 15 seconds…
- Turn to a neighbor and discuss it for a minute
Recall: A Persistence Framework

Domain Layer

Persistence Framework

- PersistenceFacade
  - get(OID, class):Object
  - put(OID, object)

- Store object in RDB
  - put(OID, Butler U.)

Relational Database

<table>
<thead>
<tr>
<th>Name</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>RHIT</td>
<td>Terre Haute</td>
</tr>
<tr>
<td>Purdue</td>
<td>W. Lafayette</td>
</tr>
<tr>
<td>Indiana U.</td>
<td>Bloomington</td>
</tr>
<tr>
<td>Butler U.</td>
<td>Indianapolis</td>
</tr>
</tbody>
</table>

University Table
Recall: Maps between Persistent Object & Database

**University Table**

<table>
<thead>
<tr>
<th>OID</th>
<th>name</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>XI001</td>
<td>RHIT</td>
<td>Terre Haute</td>
</tr>
<tr>
<td>wxx246</td>
<td>Purdue</td>
<td>W. Lafayette</td>
</tr>
<tr>
<td>xxz357</td>
<td>Indiana U.</td>
<td>Bloomington</td>
</tr>
<tr>
<td>xyz123</td>
<td>Butler U.</td>
<td>Indianapolis</td>
</tr>
</tbody>
</table>

:University

name = Butler

city = Indianapolis

oid = xyz123
Recall: Façade Design Pattern w/Brokers

PersistenceFacade
- getInstance(): PersistenceFacade
- get(OID, class): Object
- put(OID, Object)

<<interface>>
DBMapper
- get(OID): Object
- put(OID, Object)

Class

ProductSpecification
- RDBMapper
  - get(OID): Object
  - put(OID, Object)

ProductSpecification
- FlatFileMapper
  - get(OID): Object
  - put(OID, Object)

Manufacturer
- RDBMapper
  - get(OID): Object
  - put(OID, Object)

Each mapper gets and puts objects in its own unique way, depending on the kind of data store and format.
Recall: 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.
Recall Example: Template Method used for Swing GUI Framework

```java
// unvarying part of algorithm
public void update {
    clearBackground();
    // call the hook method
    paint();
}
```
It’s a bit like all you can eat fast food!
Template Method in NexGen POS 1/2

<<interface>>
DBMapper

get(OID):Object
put(OID):Object

Abstract PersistenceMapper

+get(OID):Object {leaf}
#getObjectFromStorage( ):Object

template method
hook method {abstract}
// 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
Persistence Framework

NextGen Persistence

ProductDescription
RDBMapper

SaleRDBMapper

ProductDescription
FileWithXMLMapper

ProductDescription
InMemoryTestDataMapper

Persistence

PersistenceFacade

<<interface>>

DBMapper

Abstract
RDBMapper

<<interface>>

PersistenceMapper

1

Abstract
PersistenceMapper

Abstract
RDBMapper

class
Transactional States & the State Pattern

Database transactions need:
- insert, delete, modify
- Delayed updates
  / Explicit Commits
  (rollback)

New

[new (not from DB)]

commit / insert

OldClean

[from DB]

save

rollback / reload
commit / update

OldDirty

delete

OldDelete

Deleted

commit / delete
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.
Example: State Pattern in TCP

TCPConnection

Open()  Close()  Acknowledgement()

TCPState

Open()  Close()  Acknowledgement()

TCPEstablished

Open()  Close()  Acknowledgement()

TCPListen

Open()  Close()  Acknowledgement()

TCPClosed

Open()  Close()  Acknowledgement()
State Pattern in Persistence Framework

PersistentObject

<table>
<thead>
<tr>
<th>oid: OID</th>
</tr>
</thead>
<tbody>
<tr>
<td>state: PObjectState</td>
</tr>
<tr>
<td>commit()</td>
</tr>
<tr>
<td>delete()</td>
</tr>
<tr>
<td>Rollback()</td>
</tr>
<tr>
<td>save()</td>
</tr>
<tr>
<td>setState(PObjectState)</td>
</tr>
</tbody>
</table>

PObjectState

| commit (PersistentObject obj); |
| delete (PersistentObject obj); |
| rollback (PersistentObject obj); |
| save (PersistentObject obj);   |

OldDirty State

| commit(...) |
| delete(...) |
| rollback(...) |

OldClean State

| delete(...) |
| save(...) |

New State

| commit(...) |

state \(\rightarrow\) commit( this );
Number 1555: And Some Flame Decals

This is the worst extreme sport ever

Maybe if we put on a spoiler it'll go faster.
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.
Uses for the Command Pattern

- Undo/redo
- Prioritizing and Queuing operations
- Composing multi-part operations
- Progress bars
- Macro recording
Command Pattern in NextGen POS

Transaction
commands : List
commit()
addDelete(obj:PersistentObject)
addInsert( obj:PersistentObject)
addUpdate( obj:PersistentObject)
sort()...

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

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

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

DBUpdateCommand
execute()

DBInsertCommand
execute()

DBDeleteCommand
execute()

DBCommand
object : PersistentObject
execute() {abstract}
undo() {leaf}

PersistentObject
commit()...

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

«interface» ICommand
execute( )
undo()

for each ICommand cmd cmd.execute()
Deployment Diagrams

- Recall two key Architectural views:
  - Logical Architecture
  - Deployment Architecture

- **Deployment Diagrams** provide the means to express how the physical components of the system are organized.
Outer boxes represent machines

Software artifact

Lines represent communication

Can label with protocols

Nested boxes show “execution environment nodes”
## Design Studio Calendar

<table>
<thead>
<tr>
<th></th>
<th>Monday</th>
<th>Tuesday</th>
<th>Thursday</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>8th week</strong></td>
<td></td>
<td><strong>Team 2.4</strong></td>
<td><strong>Team 2.1</strong></td>
</tr>
<tr>
<td><strong>9th week</strong></td>
<td><strong>Team 2.2</strong></td>
<td><strong>Team 2.3</strong></td>
<td><strong>Team 2.5</strong></td>
</tr>
<tr>
<td><strong>10th week</strong></td>
<td><strong>Today Team 2.4</strong></td>
<td><strong>Team 2.1</strong></td>
<td><strong>Course Wrap-up</strong></td>
</tr>
</tbody>
</table>
Homework and Milestone Reminders

- **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 2.1 Design Studio Tomorrow**

- **Reminder: Bring Laptops Tomorrow!**

- **Thursday a Project Focus Day in Class**