This page last changed on Jun 27, 2010 by smoghe.

Parking Lot - Observer

Session Objectives

By the end of this session students should ...

- Understand the differences between *polling* and *subscribe/notify*
- Understand and be able to implement the Observer pattern
- Understand the concept of *callback*
- Understand the *thinnest wire back* - the importance of sending just enough data with the notification
- Realize the trade off of breaking encapsulation (slightly) and under certain conditions in order to provide a common solution to a recurring problem
- Know the 3 questions of Observer:
  1. Who should you notify?
  2. What should you tell them?
  3. When should you tell them?

Session Overview

Activity
• Lab - Base Parking Lot • Lab - Attendant • Lab - 12 Year old Attendant • Lab - Corrupt Cop • Observer Pattern • Explore the Variations • What Direction am I Facing (Push vs. Pull)

Session Notes

Lab - Base Parking Lot

Look out the window. There's probably a parking lot there. Tell the students they are going to model a parking lot. You can park cars in a lot, until it is full to capacity and then you can no longer put any more cars in the parking lot. You can retrieve out of the parking lot which opens up space for more cars to be put in.

As a class spend 5 minutes designing. Look for a ParkingLot class. Don't need a Car class because it doesn't have any kick. Look for the first test to be parking cars until the lot is full. (Testing parking a car in a lot with spare capacity isn't good - what do you assert?).

Ask the students to implement.

Things to look for

- Tests are easy to read and express intent. Custom asserts like assertCanPark() & assertCannotPark(). Custom methods like car() & park(numberOfCars) help.
- Tests do not poke holes in ParkingLot to get information.
- Do not need Car class - just java.lang.Object
- Custom exceptions like ParkingLotFullException & UnknownCarException.

Lab - Attendant

Tell the students you have hired a parking attendant. The attendant can park and retrieve cars in multiple lots. He just parks the car in the first lot that has a space. Ask the students to implement.

Things to look for

- Attendant class
- Attendant keeps a list of lots.
- Iteration over lots to find the first full lot.

Note: Mulitple lots can be skipped until after the corrupt cop for a different flow. The flow into Composite is somewhat more natural that way, but getting the iteration out of the way early can also work well.

Lab - 12 Year old Attendant

Tell the students you actually hired your 12 year old nephew to be the attendant. It turned out that he hadn't done much driving before and scraped up several cars. You want to prevent this so you set the rule that he can't park unless the lot is less than 80% full. Ask the students to implement.

Discussion
Should ParkingLot know about the 80% limitation on attendant parking? Challenge the students on what the job of ParkingLot is. Should it really care about 80% full? Isn't that the job of the Attendant?
Discuss how the Attendant knows about ParkingLots, but we want to minimize the return relationship. This is called "the thinnest wire back" that is possible.
Ask the students how they could implement this. Through discussion *polling* and other alternatives should be explored and rejected, leaving *observation* (subscribe/notify).

Elicit real world examples of polling vs subscribe/notify:

- When baking a cake - repeatedly checking the time on your watch *vs.* setting a timer/alarm on the oven
- When roasting a turkey - repeatedly sticking a thermometer into a turkey *vs.* one of those popup turkey roasting gauges.
- To wake up at a hotel - repeatedly looking at the clock *vs.* requesting a wakeup call
  Note that media polls / elections are not a good example of our meaning of polling because they occur once and extrapolate results from a small population. Our type of polling occurs again and again on a timed basis, and does not extrapolate.

If the students aren't understanding the change from Attendant requesting information (pulling) to the ParkingLot notifying (pushing) you can use the push/pull roleplay

Give students more time to implement.

Things to look for

- ParkingLot notifying Attendant whenever a car is parked / retrieved.
- ParkingLot doesn't know anything about the 80% rule. That is calculated by Attendant based on information passed by ParkingLot (ex. capacity & number of cars).
    - Passing this information is a bit of encapsulation breakage, but the ParkingLot controls what information is sent.
- Store only available lots in Attendant
- Remove lot from available when it passes 80%. This requires passing the parkinglot in the notification.
- Re-add lot to available when it drops below 80%
- lotChanged() method on Attendant
- Notifying for change in both park and unpark methods
- Attendant constructor adds itself to parking lot
- Double rounding errors and divide by zeros
- How do you initialize the available lot list in the first place? One option is to have addListener call the notify method.

Point out that the notification is a type of *callback*. In this case the caller is the Attendant who subscribes(registers) itself with the ParkingLot. The callback is when the ParkingLot notifies the Attendant of an event. A good real world example is with phone calls: Back in the days before caller ID, when I was a poor highschool track team member and it was raining after practice, I would collect call my mom. She wouldn't accept the charges, but she would know who called because the operator told her (effectively registering me with my mom). She could then call me back to see if I needed a ride.

Lab - Corrupt Cop

Tell the students your old high school buddy Tony stops by with a proposition. He's working as a cop now and he wants to slip you some money for towing cars out of your lot. So you agree that whenever the lot has 3 or less empty spaces, you'll tell him and he can tow a car. Ask the students to implement.

Things to look for

- Cop class that is notified by the ParkingLot.
- ParkingLot tells Cop whenever a car is parked or removed. It should not know about the 3 space rule and so should not just notify when the 3 space limit is reached.
- Attendant doesn't know about Cop. (I don't want to cut my nephew in on the money).
- LotListener interface pulled up out of Cop & Attendant now that there is a second observer
- notifyLotChanged() on ParkingLot (vs. notifyListener(), etc.)
- Tests of ParkingLot notification is indirect through the Cop class tests, not directly in ParkingLot tests
- Collection of listeners on ParkingLot

Observer Pattern

Define the 3 questions of Observer

- *Who* should you notify? Those who ask (to be listeners)
- *What* should you tell them? Tell them what just happened in general terms (an event) or tell them

something significant happened, and let the listener probe for details)
- *When* should you tell them? Tell them when something interesting happens, usually in a helper method to keep code clean.

Elicit the formal description of the Observer Pattern:

Title

- Observer/Listener

Problem

- One object's behavior is closely tied to the state of another object, but we don't want to break encapsulation and poll the other objects state frequently. Reducing coupling will increase stability and reusability.

Pattern/Solution • A listener interface is created with the xxxChanged method. This interface is implemented by all listeners. • The listeners register themselves with the target at creation or during runtime. • When a specific action occurs, the target notifies its listeners by calling the xxxChanged method • Occasionally, this event will be an object itself, if it has kick • Alternately, the notification will not pass any state, but the listener will take action to obtain what it needs. Example • Event framework in Swing/other UIs. Lets us know if mouse/keyboard/etc. events occur.	!Parking Lot Observer.png!

Benefits/Liabilities

- Can be more complex to trace through an event system
- Gives support for broadcast communication
- Don't pull out interfaces if they're not needed. One Observer does not warrant an Observer interface.
- Can lead to some repeated data - e.g Attendant maintaining a list of available lots when ParkingLot has enough information to determine if parking could occur.
- Potential for circular or cascading notifications in a heavily event-based system. This problem is aggravated by simple update notifications making it difficult to deduce what event actually took place.
- Ensure subject is in a consistent state before calling notification.

Refactoring steps

1. Create a Listener interface with the xxxChanged method on it if needed
2. Implement this interface in all interested parties
3. Add an addListener type of method to the target
4. Have listeners register themselves with the target
5. Add a notify method to the target, and call it whenever relevant state is changed
6. Remove old code causing the tight coupling

Explore the Variations

- Multiple events per subject
- Observing more than one subject
- Events vs probing. Probing is when you notify something changed, but don't say what changed. The Observer must probe to find out what changed.
- If you frequently have a series of state changes occur at once and only require one final notification, then it can make sense for the client object to trigger the notification. For example through a doNotify() method.

This is a good time to dig out GOF and walk through the variations cited there. It cements the definition of pattern in their heads, and how to read GOF.

Additional Exercises

What Direction am I Facing (Push vs. Pull)

Scenario 1

- Ask for a volunteer to stand up and close their eyes
- One facilitator stands by the vocab list and points to a word
- The other facilitator asks the volunteer which word facilitator#1 is pointing to
- Allow the volunteer to ask facilitator#1 for the answer
- Change the words and ask again a few times

Scenario 2

- Now tell the volunteer they are not allowed to talk to facilitator#1
- Elicit ideas for how the volunteer can know what word facilitator#1 is pointing to. After a few ideas tell them that facilitator#1 will announce new words when he points to them
- Role play and ask the volunteer what words are being pointed to while facilitator#1.

Discussion

- Ask the students how the two scenarios differed
- Emphasize the point that the communication initiated with the volunteer in the first scenario (pull)

and with the facilitator in the second scenario (push)

- Elicit real-world examples of pushing vs. pulling
    - TV News (you don't decide what/when stories they talk about) vs. Internet news site (you pick what you read when)
    - Airlines text messaging you when your flight is delayed vs. Calling the airport/visiting their website