

Today

Welcome back! Hope vacation was relaxing...
If not, another one is only 3 weeks away...

Remaining assignment schedule:
Small HW due in 2 days (40 pts)
This gives a week to study
Last test on patterns, 1 week from Thursday (100 pts)

Let's skip ahead to Logical Architecture
(ref.: chapter 30)
This relies on the **Layers Pattern** (GoF)

Then use that as background to
Return to chapter 23 (section 23.8), and
Discuss the façade pattern (GoF)

Logical Architecture – what is it?

- Often we have emphasized the *domain*
 - ...hence de-emphasized UIs, DB access, etc.
- The domain is important to the system design
- However, there is more to system design than domain stuff
 - Other items include UI, underlying services (DB), etc.
- These items are also important software components
- See Figures 27.17 & 27.18
 - "Packages" are a way to make big modules
 - (A package contains various classes)

Logical Architecture II...

- ...is the high-level (broad, general,...) structure of the system
 - In other words, details are not part of it
- In contrast, with respect to system functionality...
 - ...the devil is in the details
 - In other words, we need to know what messages are sent to what objects, etc., etc.
- The high-level structure has less to do with function
 - Is high-level structure important even so? (why?)

Logical Architecture III

- "...everyone knows high-level structure is important...
 - so why is it important if not for functionality?"
- Let's list a few properties of software systems next that are:
 - Important
 - Not part of its functionality

Non-Functional but (Often) Important Software Properties

- Performance (10)
- Reliability (18)
- Recoverability (0)
- Usability (14)
- Portability (6)
- Maintainability (20)
- Quality
- Failsoftness (0)
- Availability (2)
- Safety (10 ½)
- Security (4)
- Expandability (2)
- Code size (3)
- ...and others

Logical Architecture in Context

- At this point we've seen 3 types of design:
 - Logical architecture (the most high-level)
 - Functional design (what we've focused on)
 - Detailed design (e.g. pseudo-code, not discussed)(These are ordered above from broad to specific)
- There are other ways to classify things
 - Logical architecture – conceptual ("logical") organization of the software
 - Vs.
 - Deployment architecture – what parts run on what processors, related network issues

Patterns that Guide *Logical Architecture*

- Recall that patterns guide how software systems are to be organized
 - **Organized = designed**, basically
- The most important *pattern* guiding *logical architecture*:
 - The **Layers Pattern**

The Layers Pattern

- Properties of *layered* organization include:
 - Each layer is a large module (e.g. a “package”)
 - Layers form a sequence in which
 - A layer requests services from one other layer
 - (the layer below it in the hierarchy)
 - A layer provides services to one other layer
 - (the layer above it in the hierarchy)
 - The layer at the top provides services to the users
 - (rather than layer, or maybe the user *is* a layer)
 - The layer at the bottom requests services from the OS
 - (not designed as part of the project, but is a layer)
 - See Figure 30.1 for example of layered organization
 - What other layers can we add to this sandwich?

The Layers Pattern II

- More properties of *layered* organization:
 - They *minimize coupling*
 - Communication is limited to neighboring layers, and
 - Kind of communication is constrained
 - How?
 - They *enhance reusability*
 - when generic services (lower layers)
 - ...are bundled into modules, and, don't call higher levels
 - They *facilitate development and maintenance*
 - ...because coupling is controlled

These are why layers are a good pattern for designing...

Layers Pattern in UML

- Layers are depicted as packages
 - (because, conceptually, they are packages)
 - (i.e., large modules containing lots of stuff)
 - See Figure 30.2
 - Figure 30.3 shows the couplings
 - Arrow points to *called* module
 - Arrows may be from/to
 - *layers*,
 - *smaller packages*, and
 - *classes/interfaces*
 - Figure 30.4 shows
 - ...the *same* couplings
 - How could you simplify it even further?
 - How much could you simplify it ultimately?

From Layers to Interaction Diagrams

- Recall two kinds of *interaction diagrams*
 - *Collaboration diagrams*
 - *Sequence diagrams*
 - Recall they say the same things but in different ways
- Interaction diagrams can reflect packages
 - ...and layers in particular
 - See Figure 30.5
 - Note that only “important” interactions are shown
 - For example, the SalesLineItem loop (where is it?)
 - These diagrams can give broad or detailed views...

Packages Vs. Subsystems

- We've discussed packages
- We've discussed subsystems
- These are not the same thing!
 - A package can group
 - related, *non-interacting* things
 - A subsystem by definition groups
 - related, *interacting* things
 - Related but not interacting – a good kind of cohesion
 - (Why?)



Layers and the Facade GoF Pattern

- Use of the layer pattern suggests
 - concurrent use of the facade pattern
- The facade pattern in a nutshell:
 - A single object provides the API
 - ...to a module containing various classes
 - It is good because changes in the module require
 - Changes to the facade object
 - **Not** changes to other code using the module



Layers and the Facade GoF Pattern II

- The facade pattern is good because
 - ...changes in the module require
 - Changes to the facade object
 - **Not** changes to other code using the module
- Suggestion: give each layer a facade object
 - ...hence, using the facade pattern
 - ...to help design using the layer pattern



Upward Collaboration with Layers

- Usually, a layer makes a request of
 - ...the next **lower** level
- Sometimes, a layer needs to influence
 - ...the next **higher** level
- Example: the clock strikes 12
 - ...time to print the daily report
 - The clock is at a lower level than the report printer
- Do this with the *observer (publish/subscribe) pattern*
 - The higher level has an event handler that runs
 - ...whenever an asynchronous event occurs at the lower level
 - What is asynchronous?



Model-View Separation Principle

- (Ref.: section 30.3)
- Domain objects should be
 - coupled as loosely as possible in general, and specifically...
 - ...to view (i.e. presentation, UI) objects
- Hence:
 - Make the presentation layer thin (does nothing but pass user inputs down a level)
 - Connect presentation and domain layers
 - ...using the observer (publish/subscribe) pattern
 - (e.g. event listeners)