

Announcements

Today:

Hand in HW

Finish Chapter 17 –

The init part of a system

The UI

...and then chapter 18 on “visibility”

small HW for Tuesday.

Will apply today’s topics

Realizing “Startup”

(Reference: Section 17.8)

- Background: we’ve “realized” the classes and most methods
 - Realize: to make real; to make into a design
- What about the initialization (startup) routine?
 - “Most, if not all, systems have a *Start Up* use case”
 - This is what executes first, to initialize the system
 - But it should be designed last!
 - Why?

Because you discover what needs to be initialized...

- by designing the rest of the functionality

How to Design the Startup

- Choose an **initial domain object**
 - This is the first domain object that will be created
 - Example (p. 270):
 - ```
public class Main {
 public static void main(String[] args){
 ProcessSaleJFrame window = new ProcessSaleJFrame();
 //initial domain object is store:
 Store store = new store();
 //store creates other domain objects:
 Register register = store.getRegister();
 //now set up user interface (UI)
```
- What was the **initial domain object**?

## How to Design the Startup - Things to Notice

- The **initial domain object** is not necessarily the “first” object in the program
  - It is the first **domain-related** object to be created
- The initialization can be done in another, non-domain object
- What is a domain object? (Non-domain object?)
- The **initial domain object** does not itself do the initialization
  - Why?
    - Rather, its capabilities are often used to help initialize
- Not all initialization is done using the **initial domain object**’s capabilities
- `main()`, or the equivalent, is...
  - ...part of the design but not part of the domain

## Startup and Interaction Diagrams

- Make an interaction diagram and send a `create()` message that creates the **initial domain object**
  - See Figure 17.17
- If the initial domain object then runs the system, send a `run()` message to it
  - Java is event-driven, so this would not be done in Java

## Startup and the POS System

- When does startup occur?

## Startup and the POS System

- Startup occurs when the POS system is run
  - A system could be configured to run POS automatically at bootstrap time
  - The first thing POS does is execute the startup
    - When might that be?
  - A system might be a normal computer and POS is run the same way a text editor might be run

## What Object Should be the *Initial Domain Object*?

- Pick an object containing most other objects
- POS system: **store** is a good choice
- For an airline reservation system, what might be a good choice?
- How about a UML assistant?

## What Should Startup() Create?

- Create store object(s)?
- Create register object(s)?
- Create productCatalog object(s)?
- Create productSpecification object(s)?
- Create sale object(s)?
- Create salesLineItem object(s)?

## What else Should Startup Do?

- It should, for example, create associations
  - E.g. productSpec objects are associated with catalog object
    - ...by setting up catalog to have various product specs as member variables (use an array?)
  - What if the number of productSpec is not known ahead of time?

## What Associations Should Startup Make?

- What associations are necessary among:
  - register object(s), productCatalog object, and productSpecification object(s)?
    - (Which need access to information in which?)
    - (...this is based on the information expert GRASP pattern)

## A Related Topic: UIs

(Reference: section 17.9)

- Applications have “logical layers”
  - See Figure 17.18
  - We’ve heard a lot about the **domain layer**
  - What about the **user interface (UI) layer**?
- Users don’t deal directly with domain objects
  - They deal with interfaces
    - ...which transmit user intentions to the domain objects

## User Interfaces II

- The startup routine can connect domain objects to UI objects
  - Java POS example:  
`ProcessSaleJFrame frame=new ProcessSaleJFrame(register);`
- Think: what UI object and domain object are connected (associated)?
- See Figure 17.18 again

## Improving the Interface

- If the UI should show a running total...
  - This is maintained by the sale object
  - UI gets it by sending a message to sale
  - See Figure 17.19 (how would you fix it?)
  - (Is it fixed in the new edition?)
  - Maybe this should be an Extra Credit in-class exercise...

## Interface – Additional Notes

- Note in Figure 17.19, improved, that
  - UI does no data crunching
  - This is consistent with **UI design** principles
  - See box above Fig. 17.19

## New Topic: Visibility

- Reference: Chapter 18
- There are four ways objects can be visible to each other
- Designs can use any of these

## 1. Attribute Visibility

- This simply means one object is an attribute of another:
- public class Register{
  - ...  
private ProductCatalog catalog;  
...  
} //see p. 281



## 2. Parameter Visibility

- When one object is passed as a parameter to a method of another
  - ...they have parameter visibility
- Example (Fig. 18.3, p. 282):
  - A sale object gets a makeLineItem message, with a ProductSpec passed as a parameter
- What is the method, where is it, who calls it, and what is its argument?



## 3. Local Visibility

- Occurs when an object is created or returned, and stored *locally in a method*
- This is usually more temporary a storage method than attribute visibility
  - Why?



## 4. Global Visibility

- Global data (data visible everywhere in the system)...
  - ...has global visibility
- Java doesn't even have this kind!
- Why is it so dangerous that Java doesn't have it?