

CSE70: Lecture 8

- Project last release started today!
- Tuesday Lab
 - Look at the Checklist
- Wednesday
 - Demo
- Today
 - Refactoring example
 - Patterns
 - Architectural styles
 - Design examples
 - The Cathedral and the Bazaar

Demos

- Projects due at 11:59am in repository
- From the repository
 - jar of the latest version
 - on my laptop
 - If you have special requirements please let me know in advance
- You are not required to make slides
 - But please do if you like
- Demos will be approx. 15 minutes long
 - Show off the implemented features
 - Gives us some technical details
 - what was challenging?
 - interesting design decisions?
- Every member of the team can speak
- Presence is mandatory

Refactoring

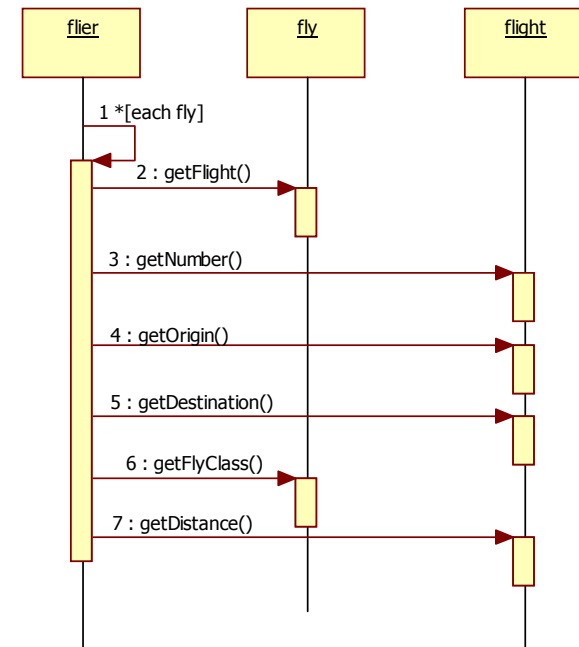
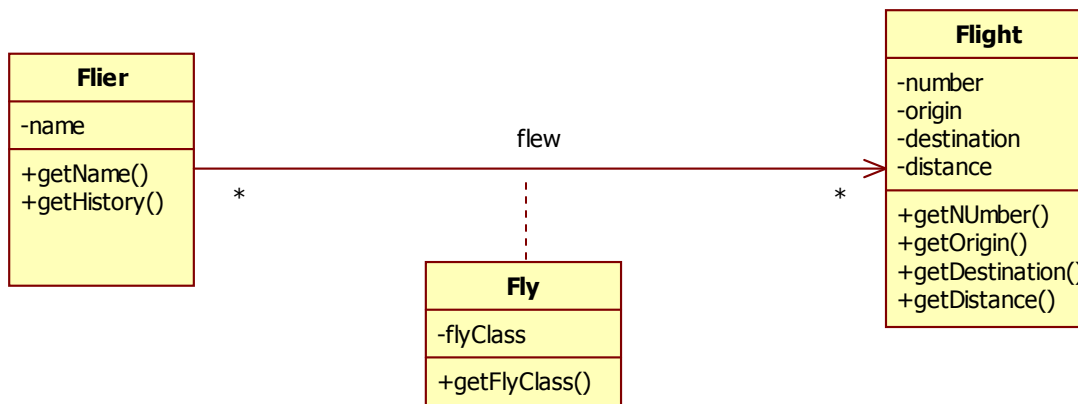
- Why?
- When?
- What?

Example

name's fly history

origin destination

Total frequent flier miles



Design Patterns

- Christopher Alexander
 - There is an objective measure for quality of architectural design
 - What is the problem to solve and the context
- *Patterns*
 - A solution to a problem in a context
- Patterns in software design
- *Design Patterns: Elements of Reusable Object-Oriented Software*
 - Gamma, Helm, Johnson, and Vlissides (GoF)

Pattern

- Why study patterns?
- Name, Classification, Intent, Solution, ...
- Categories
 - Creational
 - Structural
 - Behavioral
 - Concurrency

Singleton

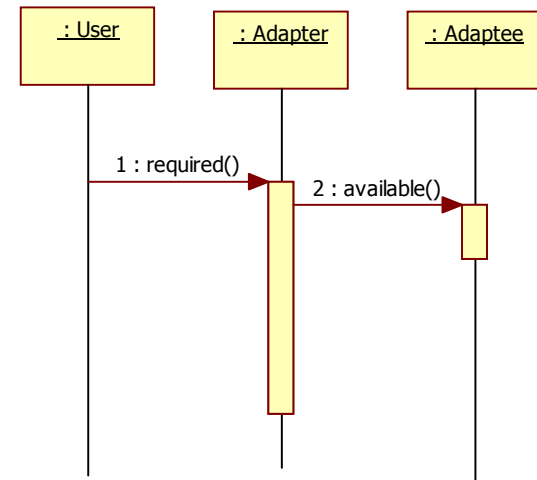
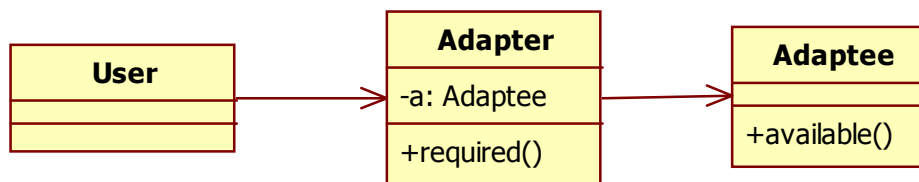
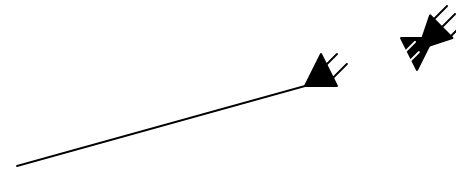
- Create one and only one instance

```
public class Singleton {  
  
    private static Singleton instance = new Singleton();  
  
    private Singleton() {}  
  
    public static Singleton getInstance() {  
        return instance;  
    }  
  
}
```

Singleton
<u>-instance</u>
-Singleton() <u>+getInstance(): Singleton</u>

Adapter

- Adapt an interface

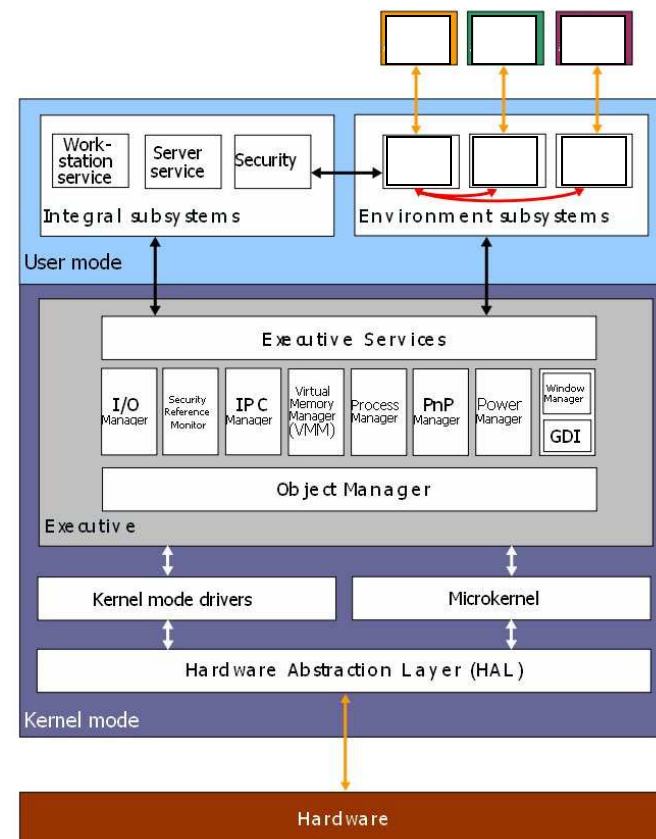
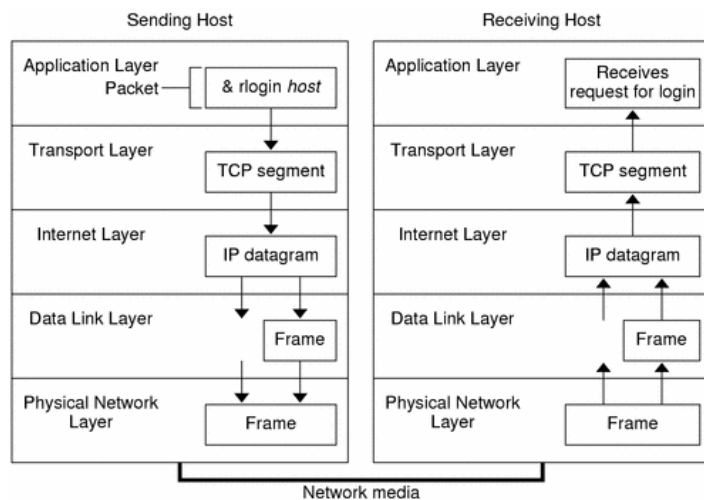


Behavioral & Concurrency

- Iterator
 - Iterates through element of an aggregation
 - Implementation independent
- Thread Pool
 - Maintain a pool of threads
 - Assign work to thread dynamically

Architectural Styles

- High-level models
 - Describe components and their interactions
- Layered architecture
 - Stack of interfaces
 - Bottom up construction



Architectural Styles

- Layered architecture
 - + Cohesive interface
 - + Low coupling
 - + Information hiding
 - + Modularity
 - + Incremental development
 - Imposes indirection

Architectural style

- Filter: component that transforms data
- Pipes: support data flow connecting filters
- Pipe-and-Filter
 - + Low coupling
 - + Information hiding
 - + Modularity
 - + Incremental development
 - Doesn't reflect well interactions and bidirectional communication
 - The interface must be minimal
 - Some processing might require the entire input

Architectural style

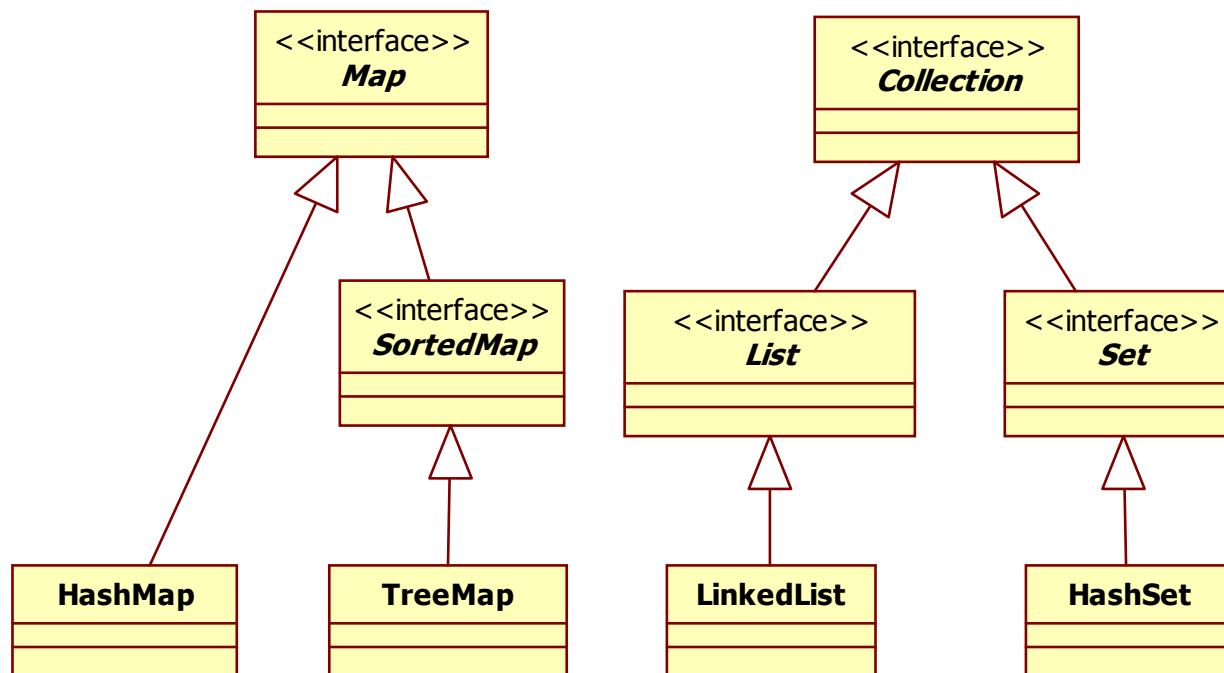
- Model: domain model and data
- View: user interface
- Controller: command execution
- MVC
 - + Modularity
 - + Incremental development
 - Components might be View and Controller
 - Coupling?

Architectural style

- Event: observable change in state
- Dispatcher: notifies registered observers
- Event-Driven
 - + Low coupling
 - + Information hiding
 - + Modularity
 - + Incremental development
 - Dispatcher might become a bottleneck

Examples

- The collection framework (Java)



Examples

- GUIs with Swing (java)
 - MVC + Event-Driven
 - EventObject
 - *getSource()*
 - *toString()*



Figure from <http://java.sun.com/docs/books/tutorial/uiswing/events/intro.html>

The Cathedral and the Bazaar

- Cathedral
 - Centralized approach
 - Individuals/Small group
 - Late release
- Bazaar
 - Release early and often
 - Open development

The Cathedral and the Bazaar

- Lessons
 - 1.Motivation
 - 2.Reuse
 - 3.“Plan to throw one away; you will anyhow”
(Fred Brooks, *The Mythical Man-Month*)
 - 6.Users/Hackers
 - 7.Release
 - 8.Bugs become shallow

The Cathedral and the Bazaar

- Linus's Law
- In the cathedral
 - Development problems and bugs are deep phenomena
 - Search for stability
- In the bazaar
 - Bugs are/become shallow phenomena
 - Debugging is parallelizable
 - 2 Linux versions available

How many eyeballs tame complexity

- non-source-aware users bug reports
 - Omit critical data
 - Hard to reproduce
- source-aware users bug reports
 - source-code level characterizations
- Brook's Law
 - “Adding more programmers to a late project makes it later”

Applying the bazaar style

- Some more lessons
 - 9. Smart data, dumb code
 - 10. Beta-testers are the most valuable resource
 - 12. Envision solutions means reconsider the problem
- “La perfection est réalisée, pas quand il n'y a rien à davantage ajouter, mais quand il n'y a plus rien à emporter.”
 - Antoine de Saint-Exupéry

Summary

- Today
 - Design Patterns
 - Architectural styles
 - The Cathedral and the Bazaar
- Next time
 - Tuesday we are in the lab
 - Wednesday here for the demos
 - Thursday?