# System Design

The big picture

# System Design

- Have the model, but the model needs help...
  - What services is required by the model?
  - MP: Have event bus and GUI, probably need a few others
- During system design we try to
  - Identify services (implemented as subsystems)
    - Mandatory to have a least one
  - Manage overall structure and flow
  - Global design issues
  - Resource handling (images, etc.)

# System Design Overview

- Global design decisions
- Software decomposition (the pieces)
  - Tiers, subsystems, interfaces
- Layering (overall)
- Communication
- More dependency analysis
- Persistency, storing data, data formats
- Concurrency issues
- Security
- Boundary conditions; Start, stop, errors
- Selecting platform, **done**, (Java SE >= 1.6 )

# Global design decisions

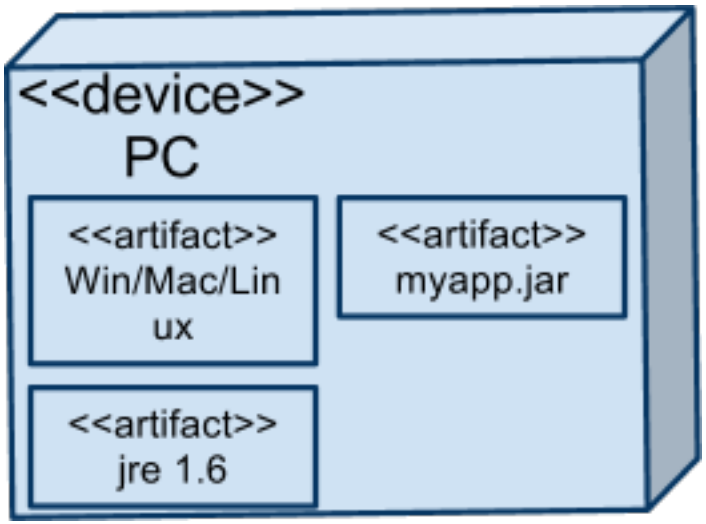
- Decisions affecting "everything"
  - Distributed application (optional). Where does the model live?
  - Globally unique id's (the spaces in MP?...)
  - Global data structures (accessible globally)
  - MVC model, **done**
  - Life cycles of objects (possible to restore game??...)
  - Interoperability requirements
  - Communication (also inside single application), we have the EventBus, is that enough?
  - ...

# Subsystems (services)

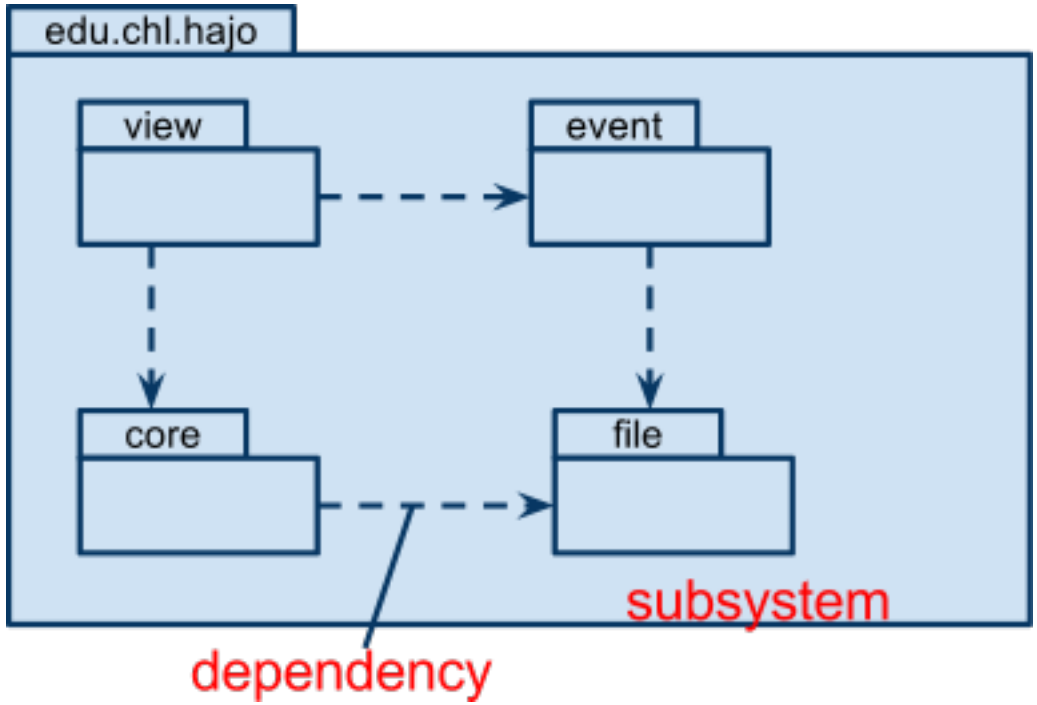
- Some typical
  - Persistence (store/retreive)
  - Printing
  - Communication
  - Rule systems (business/game rules)
  - Engines, simulation engine
  - Graphics 2D, 3D...
  - Processors (text  formatter, spell checker)
  - Security, authorization module
  - Mappers, mapping between formats

# UML for System Design

- Deployment diagram (left)
- Package diagram (right)
- Also: Class diagrams




myapp.jar



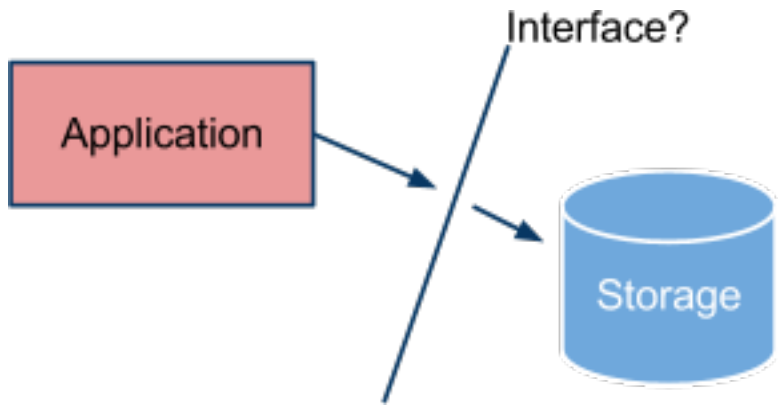
# "In house" or Not?

- Typically you don't implement subsystems for
  - Graphics
  - Sound
  - Data handling, XML, ...
  - Networking
  - ... find somewhere!
- Always look for high level
  - Network: Sockets, **NO!!!!**
    - XML-RPC, RMI, ... other,  ... much better
  - Databases to Objects, very hard, use JPA, Hibernate, ...
- If no other possibility ... in-house (avoid)
  - Possibility: Adapt existing code (Adapter pattern)

# Subsystem Interfaces

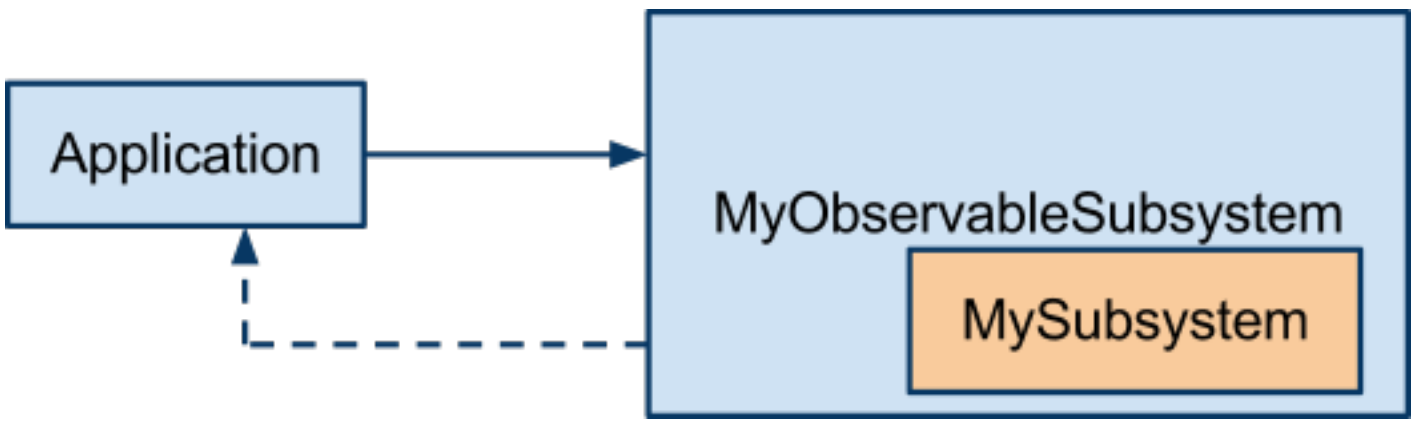
- The interfaces are the important design decisions
  - Subsystem of course has a single responibility
  - Should be possible to swap implementations

- Example: IPersistency.java
  - Interface to storage system
    1. What would you like to do (not how)?
    2. Implement it: Flat files, serialization, XML, real database



# Subsystem Implementation

- Prefer stateless subsystems
- Standard: Facade + Factory design pattern
  - Factory always return interface type
- Possibly add features by wrapping (Decorator pattern)
  - Make it a singleton
  - Make it observable
  - ...



# Testing of Subsystems

- Any in-house subsystem  is of course thoroughly tested before attached to the application
  - Hopefully others are too

# Keep the Model Clean

- Again: Minimize use of foreign (service) code in model
- Let model use subsystems in a disciplined manner
- Preferable: Only one model object uses any given subsystem

# Lookups

- Very common need for global lookup
  - Singletons
  - Resource Locator
    - Singleton with methods to locate objects
    - Read only
  - Global maps
    - Enum as keys (no misspelling)
    - Read only

# Resources

- How to find/organize resources?
  - Use Resource Bundles (see demos)
    - java.util.ResourceBundle
    - A map as a text file. File automatically read and converted to Java "map-like" object
  - For images use ClassLoader class
    - getResource(s), findResource(), ...
  - Possible XML, use Java JAXP (API For XML processing)
    - Even better (simpler) XStream library, see sample on course page.

# Wiring It Togheter

- Where and when to wire together the application?
  - Static wiring; fixed references
  - Dynamic wiring; changing references
- Ad-hoc (non general)
  - Class "A" creates "B" creates "C", ...
    - Creation all over!
    - Dependencies..?!
- Centralized creation better
  - If simple, create/wire in Main class
  - Else, Builder design pattern or similar
- Possibly (advanced) use a framework
  - A framework can "inject" objects into other objects
  - Very loose coupling
  - Have look at testweld.ep, testguice.ep (on course page)
- **Note:** Interfaces never have methods for creation

# Summary

- One weakness with our process is the lack of **design upfront**, possible problems at the system level
- If a bit more experienced we should have worked with system design in parallel

Next: No... this is the ending.  Thanks and GOOD LUCK!!