System Design

The big picture

System Design

- During system design we try to create the overall structure
  - Partitioning (divide) the problem and the system to be built into discrete pieces
    - Have core and GUI, need possible many supporting subsystems
  - Create interfaces between these pieces
  - Manage overall structure and flow

System Design Overview

- Design goals
- 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

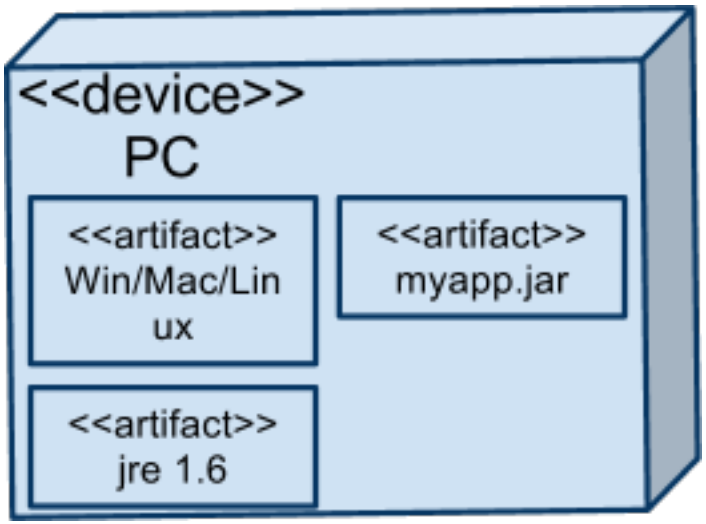
- Decisions affecting "everything"
  - Distributed application (optional)
  - Globally unique id's
  - Global data structures (accessible globally)
  - MVC model, **done**
  - Life cycles of objects
  - Interoperability requirements
  - Communication (also inside single application), we have the EventBus,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

Subsystems

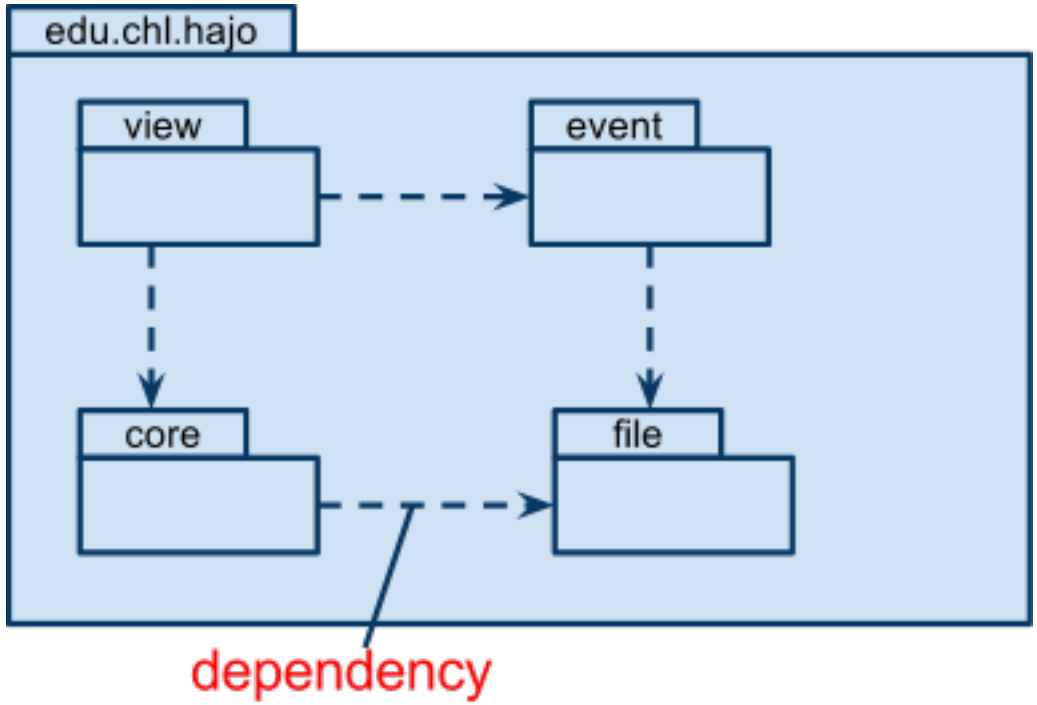
- Some typical
  - Persistence (store/retreive)
  - Printing
  - Communication
  - Rule systems (business/game rules)
  - Engines, simulation engine
  - Processors (text formatter, spell checker)
  - Security, authorization module
  - Mappers, mapping between formats

UML for System Design

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



myapp.jar

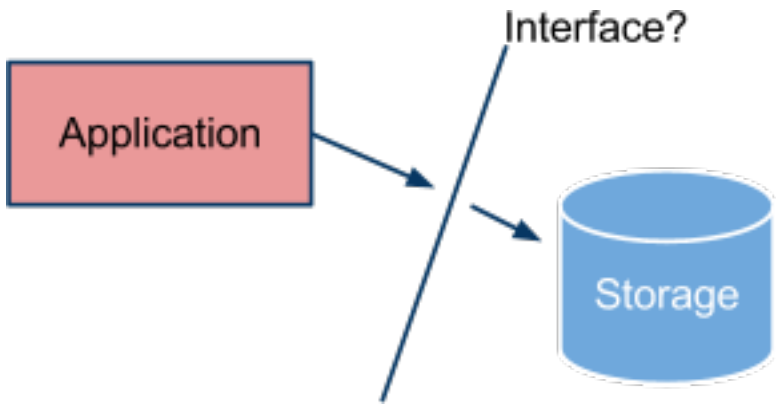


Subsystem to Search For

- 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, ... probably better
  - Databases to Objects, very hard, use JPA, Hibernate, ...
- ... find, possible wrap using Adapter DP to match our needs (What interface do we have? What do we need?)

Subsystem Interfaces

- Once again: The interfaces are the important design decisions, exact implementation is a **detail**!.
- Example: IPersistency
  - Interface to storage system
    1. What would you like to do?
    2. Implementation: Flat files, serialization, XML, real database

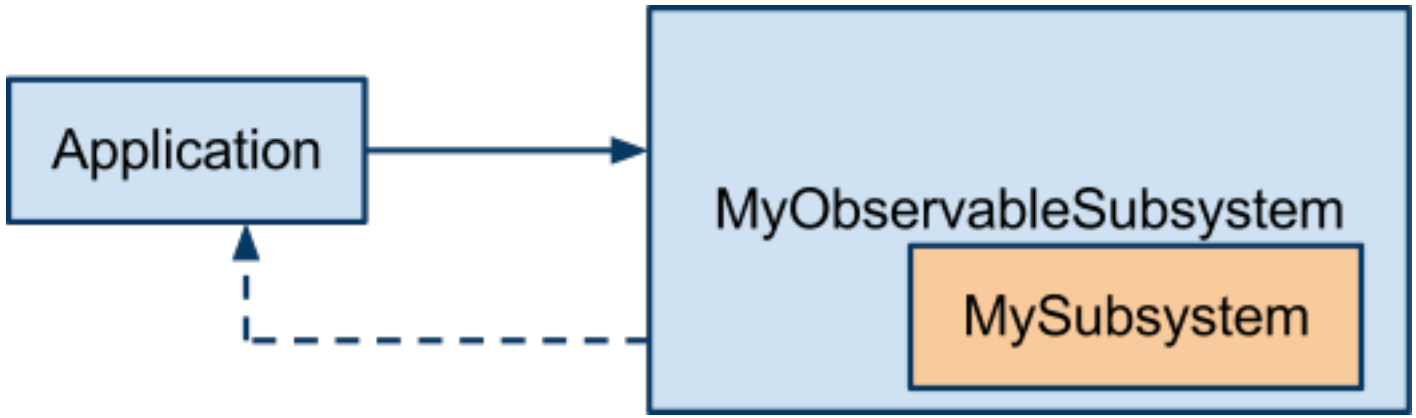


Subsystem In House

- "In house" (code yourself)
  - In house implementation
    - Standard: Facade + Factory method (DP)
    - Interface to subsystem delivered by factory method

Subsystem Implementation

- Subsystem has single responsibility
- Possibly add features by wrapping (decorator pattern)
  - Make it a singleton
  - Make it observable
  - ...



Testing of Subsystems

- Any subsystem is of course thoroughly tested before attached to the application

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
  - 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)

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

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!!