Actor-Based Design
Guidelines for creating actor-based designs.

- What are some of the design heuristics, or best practices when using actors in the implementation of concurrent system?
Like most techniques, actors are not a golden hammer, appropriate for all situations.

- Sending messages incurs overhead.
  - Heavy interaction between actors
  - Request and response messages
  - Voting or quorum of actors

- Preferred characteristics
  - Independent actors
  - "Fire-and-forget" interactions
  - Asynchronous I/O

Not good for actors.
The design task is to decompose the system into actors and the messages communicated.

- Actors may be the easiest to identify
  - You are the director of a play; What are the clear and distinct roles. *(Separation of concerns)*
  - Hierarchical structure is often used.
    - Supervisor actors
  - Active objects in thread-based designs are not necessarily the best actors.
  - Consider Typed Actors to convert a POJC application into an message based one.

- You need to identify actor responsibilities
  - Pick a well-defined, limited set for each actor *(Cohesion)*
  - Responsibilities as independent as possible *(Coupling)*
Messages are the next stage of design.

- Messaging is the lifeblood of an actor-based system.
  - *What is the minimum amount of interaction needed between actors?*
  - *What messages would result?*
  - *Send only immutable messages*
  - *Decide on the message content*

- Initially, put concerns of overhead aside.
  - *Don't be afraid to pass immutable data around.*
  - *The more you can do with messages, the less you have to worry about synchronization.*
  - *Today's systems can handle a lot.*
  - *“Make it work, make it right, …”*
This sounds very much like an object-oriented design decomposition.

- In some ways, actors are the ultimate objects.
- Just like designing OO systems, the static part (class diagrams) is easy.
  - **Identifying actors and messages says nothing about the dynamic operation.**
  - **Run feature scenarios through your system to discover design gaps, or awkward interactions.**
Now, you can consider some of the performance issues that might arise.

- **Use actors for prototyping**
  - Prototype the concurrent solution using a purely Actor-based design.
  - Use profiling tools to identify parts of your application that might benefit from a different approach.
  - “Make it work, make it right, make it fast.”

- **Watch out for I/O**
  - Actors and I/O should be interleaved carefully.
  - Asynchronous I/O and actors fit well together.
  - Blocking I/O can cause an actor to starve other actors.
  - Dead lock (and dead actors) cause problems.
The Advantages of Messaging

- “Messaging systems are an abstraction on a synchronous process”
- Actors only communicate to the outside world by sending & receiving messages:
  - Messages are maintained in a mailbox until the actor retrieves them (no need to maintain a separate message queue)
  - Only one message is handled at a time in single-threaded fashion, so state can be maintained without explicit locks.
  - Each actor stands alone!