Basic Principles:
- Avoid the UI Layer where possible
- Break down UI components into smallest units possible

Model View Presenter
- Extremely thin layer
- Each method should be one line of delegation
- Expose callbacks to register event listeners
- Maintain state about view
- Understand which views get updated
- Only interact with a view of an interface
- Do most of the work
- Is used to tie the View and Presenters together
- Threading typically handled here
- Assemble views into the final layout.
- Externalised from each view

Acceptance Test Patterns
- Fixtures
  - Describe each significant screen (LoginDialog) with things you can do on each screen.
  - Use terms that the business uses to describe the things on the screen
  - Implemented using libraries internally to find existing swing components
- Threading
  - Wrap SwingUtilities in an interface (EventQueue) and another implementation can be used in unit tests to provide consistently predictable results
- Be Careful Of
  - Swing having side effects – frames are not disposed until we stop the JVM – so state may be inconsistent until the application finishes

Test Driven Development
- Acceptance Tests (1) may spawn multiple unit integration tests and will spawn multiple unit tests
- Use the Acceptance Test to drive out other tests
- Split types of tests physically as well as run them separately (faster running ones first)
- Requires high discipline
- Driven by a User Story or a Use Case
- Refactor to make code more readable and draw out other design patterns
- Takes practice

Tips
- Small views are better
- Name every important UI component
- Need several types of groups of tests (acceptance or end-to-end, unit integration, and pure unit tests)
- Layout generally not worth automated regression testing