HW Assignment #2

Ria Singh & Anika Bhatnagar

Problem 5.1, Stephens page 116

What's the difference between a component-based architecture and a service-oriented architecture?

A component-based architecture structures software as a collection of reusable, self-contained components that interact with each other via well-defined interfaces.

- Components are modular, meaning they can be developed, tested, and replaced independently.
- Typically deployed within the same application or system.
- Components communicate using direct method calls.

Problem 5.2, Stephens page 116

Suppose you're building a phone application that lets you play tic-tac-toe against a simple computer opponent. It will display high scores stored on the phone, not in an external database. Which architectures would be most appropriate and why?

Model-View-Controller (MVC)

- Separation of Concerns: The game logic (Model), user interface (View), and input handling (Controller) are clearly separated, making the app easier to maintain and expand.
- Simplicity: MVC is a common pattern in mobile development (e.g., SwiftUI, Jetpack Compose).
- Scalability: If you later want to add features (e.g., online multiplayer), the architecture supports easy modifications.

Component-Based Architecture

- Reusability: Components such as the game logic, AI opponent, and UI elements can be developed separately.
- Maintainability: If you later want to improve the AI or change the UI, you can do so without breaking other parts of the app.

Event-Driven Architecture

- If you implement features like animations, move history, or undo, an event-driven approach would help manage user interactions efficiently.
- An event triggers when a player moves, updating the board and checking for a win.

Problem 5.4, Stephens page 116

Repeat question 3 [after thinking about it; it repeats question 2 for a chess game] assuming the chess program lets two users play against each other over an Internet connection.

When implementing an internet-based chess game where two players compete remotely, the system requires additional features beyond local play. These include:

- Networking Support: A client-server model using protocols such as WebSockets or TCP/IP to handle real-time communication.
- Move Validation & Synchronization: Ensuring both clients receive valid board updates simultaneously.
- User Authentication & Session Management: Allowing players to reconnect if they disconnect mid-game.
- Latency Handling: Addressing network delays through predictive move handling or timeout mechanisms.
- Security: Preventing cheating via encrypted move transmission and validation on the server side.

This expands on local play requirements by introducing network reliability, security, and real-time synchronization concerns.

Problem 5.6, Stephens page 116

What kind of database structure and maintenance should the `ClassyDraw` application use?

The *ClassyDraw* application, which likely manages and stores user-generated graphical elements, should use a relational database for structured storage or a document-based NoSQL database for flexibility.

Recommended Database Structure:

- Tables (for relational DB like MySQL, PostgreSQL)
    - Shapes Table: Stores shape type (Line, Rectangle, Ellipse, Star, Text) and general properties (position, size, color).
    - Text Table (if applicable): Stores text content separately with a reference to the shape ID.

- User Table: If multi-user, stores accounts and permissions.

Maintenance Considerations:

- Indexing: Optimize retrieval for large drawings.
- Backup Strategies: Ensure regular backups to prevent data loss.
- Concurrency Handling: Manage simultaneous edits if collaboration features exist.

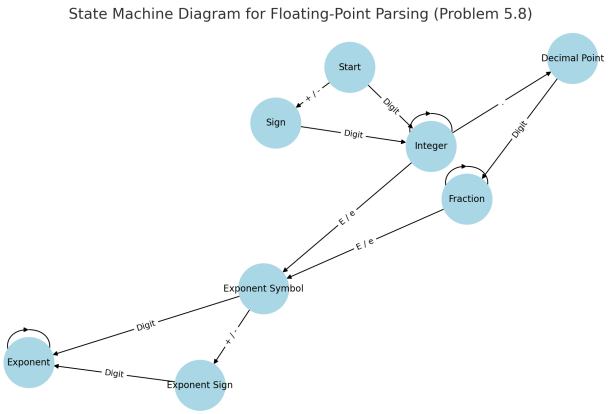
A NoSQL solution (e.g., MongoDB) could store each drawing as a document containing all necessary properties.

Problem 5.8, Stephens page 116

Draw a state machine diagram to let a program read floating point numbers in scientific notation as in +37 or -12.3e+17 (which means -12.3 x 10^{17}). Allow both E and e for the exponent symbol. [Jeez, is this like Dr. Dorin's DFAs, or *what*???]

A finite state machine (FSM) for reading floating-point numbers in scientific notation must handle:

1. Sign Parsing (`+` or `-`)
2. Integer Parsing (digits before the decimal)
3. Decimal Parsing (`.` followed by digits)
4. Exponent Symbol (`E` or `e`)
5. Exponent Sign (`+` or `-` after `E/e`)
6. Exponent Digits



Problem 6.1, Stephens page 138

Consider the `ClassyDraw` classes `Line`, `Rectangle`, `Ellipse`, `Star`, and `Text`. What properties do these classes all share? What properties do they not share? Are there any properties shared by some classes and not others? Where should the shared and nonshared properties be implemented?

All *ClassyDraw* shapes (Line, Rectangle, Ellipse, Star, and Text) share common properties:

- Shared Properties (Implement in a Drawable parent class):
    - Position (X, Y)
    - Stroke color and width
    - Fill color (for closed shapes)
- Unique Properties (Implement in individual subclasses):
    - Line: Start and end points
    - Rectangle: Width, height
    - Ellipse: Major and minor axes
    - Star: Number of points
    - Text: Font, size, content

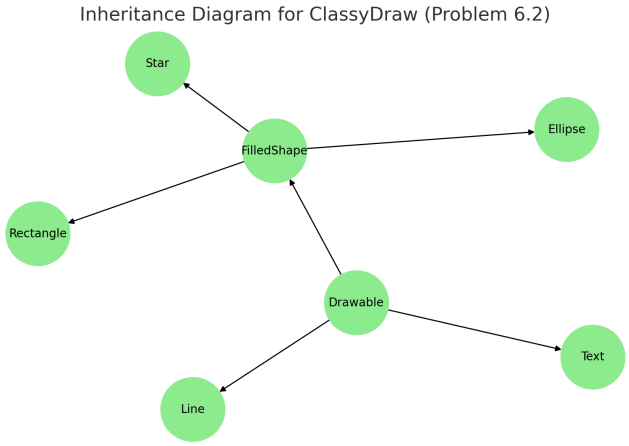
Some properties, like fill color, apply only to closed shapes (Rectangle, Ellipse, Star) and should be implemented in an intermediate FilledShape class.

Problem 6.2, Stephens page 138

Draw an inheritance diagram showing the properties you identified for Exercise 1. (Create parent classes as needed, and don't forget the **`Drawable`** class at the top.)

The provided business classes share common attributes (Name, Phone, Address), suggesting an abstract superclass Person.

- Employee extends Person, containing EmployeeID.
- Customer and Supplier extend Person, with additional attributes for billing or products.
- Manager and VicePresident extend Employee, including management-related fields.



Problem 6.3, Stephens page 139

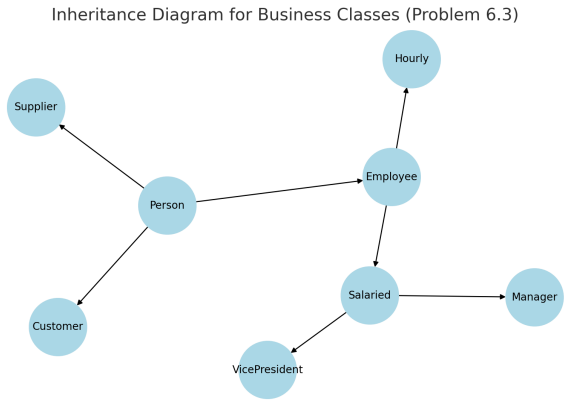
The following list gives the properties of several business-oriented classes.

- Customer — Name, Phone, Address, BillingAddress, CustomerID
- Hourly — Name, Phone, Address, EmployeeID, HourlyRate
- Manager — Name, Phone, Address, EmployeeID, Office, Salary, Boss, Employees
- Salaried — Name, Phone, Address, EmployeeID, Office, Salary, Boss
- Supplier — Name, Phone, Address, Products, SupplierID
- VicePresident — Name, Phone, Address, EmployeeID, Office, Salary, Managers

Assuming a `Supplier` is someone who supplies products for your business, draw an inheritance diagram showing the relationships among these classes. (Hint: Add extra classes if necessary.)

The provided business classes share common attributes (Name, Phone, Address), suggesting an abstract superclass Person.

- Employee extends Person, containing EmployeeID.
- Customer and Supplier extend Person, with additional attributes for billing or products.
- Manager and VicePresident extend Employee, including management-related fields.



Problem 6.6, Stephens page 139

Suppose your company has many managerial types such as department manager, project manager, and division manager. You also have multiple levels of vice president, some of whom report to other manager types. How could you combine the **`Salaried`**, **`Manager`**, and **`VicePresident`** types you used in Exercise 3? Draw the new inheritance hierarchy.

To better structure managerial roles:

1. Combine Salaried, Manager, and VicePresident into an Employee hierarchy.
2. Introduce intermediate classes for managerial roles:
    - Manager → General class for department/project managers
    - SeniorManager → Includes division managers
    - Executive → Covers all vice presidents
3. Use interfaces for cross-reporting relationships (e.g., ManagesEmployees).