CMSI 4072: Senior Project II Written Homework Assignment 01

Problem 5.1, Stephens page 116: What's the difference between a component-based architecture and a service-oriented architecture?

Scope and Granularity: Components are often finer-grained and used within a single application, while services are coarser-grained and can be accessed across multiple applications or systems.

Communication: Components interact through direct method calls within the same process, whereas services communicate via network calls, often using standardized web protocols.

State Management: Components may maintain state within an application, while services are typically stateless, handling each request independently.

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?

Component-Based Architecture (CBA):

- Modularity: The application can be divided into distinct components, such as the user interface, game logic, and data storage. This separation allows for easier development and maintenance.
- Reusability: Components like the game engine or AI opponent can be reused in other projects or games, promoting efficient code utilization.
- Simplified Communication: Since all components reside within the same application, they can interact directly without the need for complex communication protocols.

Why Not Service-Oriented Architecture (SOA)?

- Overhead: SOA involves creating independent services that communicate over a network, which introduces unnecessary complexity for a simple, self-contained application.

- Distributed Systems: SOA is designed for applications that require integration across different systems or platforms, which isn't the case for this standalone mobile game.

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.

Service-Oriented Architecture (SOA):

- Distributed Functionality: SOA allows the game's functionalities—such as matchmaking, game state management, and move validation—to be implemented as independent services. These services can operate on different servers, facilitating scalability and maintainability.
- Interoperability: By utilizing standard communication protocols (e.g., HTTP, WebSocket), SOA enables different parts of the system to interact seamlessly, regardless of the technologies used in each service. This is crucial for supporting various client platforms (e.g., web browsers, mobile devices).
- Loose Coupling: Services in SOA are designed to be independent, meaning changes in one service (like updating the matchmaking algorithm) have minimal impact on others. This modularity simplifies updates and maintenance.

Why Not Component-Based Architecture (CBA)?

- Tight Coupling: CBA involves integrating components within a single application, which can lead to tight coupling. In an online chess game, this could make it challenging to manage distributed functionalities like real-time player interactions and game state synchronization.
- Scalability Limitations: As the user base grows, a component-based system may face difficulties scaling efficiently, whereas SOA's distributed nature inherently supports scalability.

Problem 5.6, Stephens page 116: What kind of database structure and maintenance should the **ClassyDraw** application use?

Database Structure:

1. Entities and Relationships:
    - Diagrams Table: Stores metadata about each diagram (e.g., diagram ID, name, creation date, last modified date).
    - Elements Table: Contains details of individual elements within diagrams (e.g., element ID, type [class, interface, etc.], name, properties).
    - Relationships Table: Captures associations between elements (e.g., relationship ID, type [inheritance, association], source element ID, target element ID).
2. Normalization:
    - Third Normal Form (3NF): Ensure that the database schema is normalized to 3NF to eliminate redundancy and maintain data integrity.
3. Indexing:
    - Primary and Foreign Keys: Establish appropriate keys to maintain relationships between tables.
    - Search Optimization: Index columns frequently used in search queries, such as diagram names or element types, to enhance performance.

Database Maintenance:

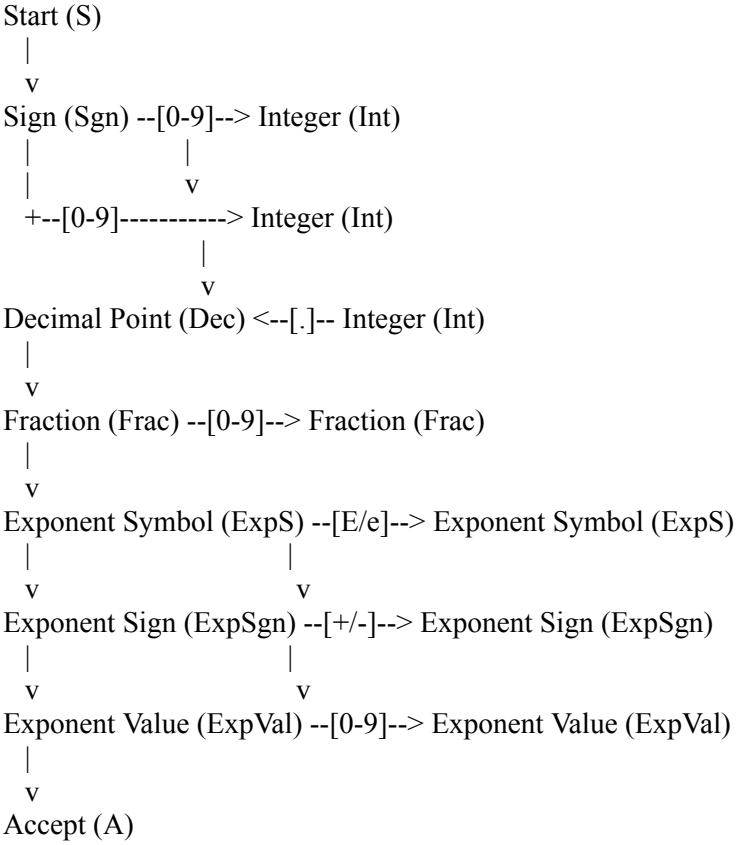
1. Regular Backups:
    - Automated Backup Schedule: Implement regular backups to prevent data loss, especially before and after significant updates or maintenance activities.
2. Data Integrity Checks:

- Validation Routines: Regularly run scripts to check for orphaned records, broken relationships, or inconsistent data entries.

3. Performance Monitoring:
    - Query Optimization: Analyze and optimize slow-running queries.
    - Resource Utilization: Monitor database performance metrics to identify and address bottlenecks.
4. Security Measures:
    - Access Controls: Define user roles and permissions to restrict unauthorized data access.
    - Encryption: Encrypt sensitive data, both at rest and in transit, to protect against breaches.

By implementing a well-structured database schema and adhering to regular maintenance practices, the ClassyDraw application can ensure data integrity, optimal performance, and scalability.

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 1017). Allow both E and e for the exponent symbol. [Jeez, is this like Dr. Dorin's DFAs, or *what*???]



Problem 6.1, Stephens page 138: Consider the **ClassyDraw** classes **Line**, **Rectangle**, **Ellipse**, **Star**, and **Text**.

1. What properties do these classes all share?
2. What properties do they **NOT** share?
3. Are there any properties shared by some classes and not others?
4. Where should the shared and nonshared properties be implemented?

Shared Properties:

All these classes represent drawable objects and share common properties related to their visual representation and positioning:

- Positioning:

    - UpperLeft (X, Y): Coordinates specifying the top-left corner of the object.
    - Width and Height: Dimensions of the object.
- Appearance:
    - ForegroundColor: Color of the object's outline or text.
    - BackgroundColor: Fill color for shapes that support filling.
    - LineThickness: Thickness of the object's outline.
    - DashStyle: Pattern of the outline (solid, dashed, etc.).

Non-Shared Properties:

Certain properties are unique to specific classes:

- Text Class:
    - Font: Specifies the font type for the text.
    - Content: The actual string of text to be displayed.
- Shape Classes (Rectangle, Ellipse, Star):
    - FillColor: Color used to fill the interior of the shape.
- Line Class:
    - StartPoint and EndPoint: Coordinates defining the line's endpoints.

Properties Shared by Some Classes:

Some properties are common to multiple, but not all, classes:

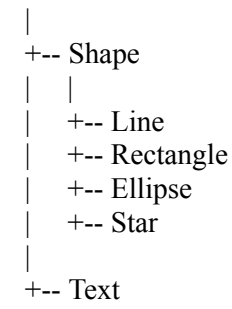
- FillColor: Applicable to shapes like Rectangle, Ellipse, and Star, but not to Line or Text.

Implementation Strategy:

To manage these properties efficiently, an inheritance hierarchy can be established:

1. Drawable (Base Class):
    - Defines properties common to all drawable objects: UpperLeft, Width, Height, ForegroundColor, BackgroundColor, LineThickness, and DashStyle.
2. Shape (Intermediate Class):
    - Inherits from Drawable.
    - Adds properties specific to fillable shapes: FillColor.
3. Specific Shape Classes:
    - Rectangle, Ellipse, Star: Inherit from Shape and implement any additional properties or methods unique to each shape.
    - Line: Inherits directly from Drawable and includes properties like StartPoint and EndPoint.
    - Text: Inherits from Drawable and adds properties such as Font and Content.

Problem 6.2, Stephens page 138: Draw an inheritance diagram showing the properties you identified for Exercise 6.1. [Create parent classes as needed, and don't forget the **Drawable** class at the top.]



- The Drawable class encapsulates properties and behaviors common to all drawable objects, such as color, position, visibility, and rotation. This promotes code reuse and ensures consistency across all classes
- The Shape class groups properties shared by geometric shapes (lineWidth, fillColor, borderStyle). This avoids redundancy in the Line, Rectangle, Ellipse, and Star classes
- The Text class has unique properties (content, fontSize, fontFamily) that are not shared with geometric shapes. Therefore, it doesn’t need to inherit from Shape