UML Usage in Software Development

CSC3350

Dr. William Greg Johnson
Department of Computer Science
Georgia State University

Georgia State University | COLLEGE OF ARTS & SCIENCES

Understanding UML: A Visual Language for Software Design

- Explore the power of UML diagrams as a standardized visual language for software development.
- Key diagram types.
- Essential best practices.

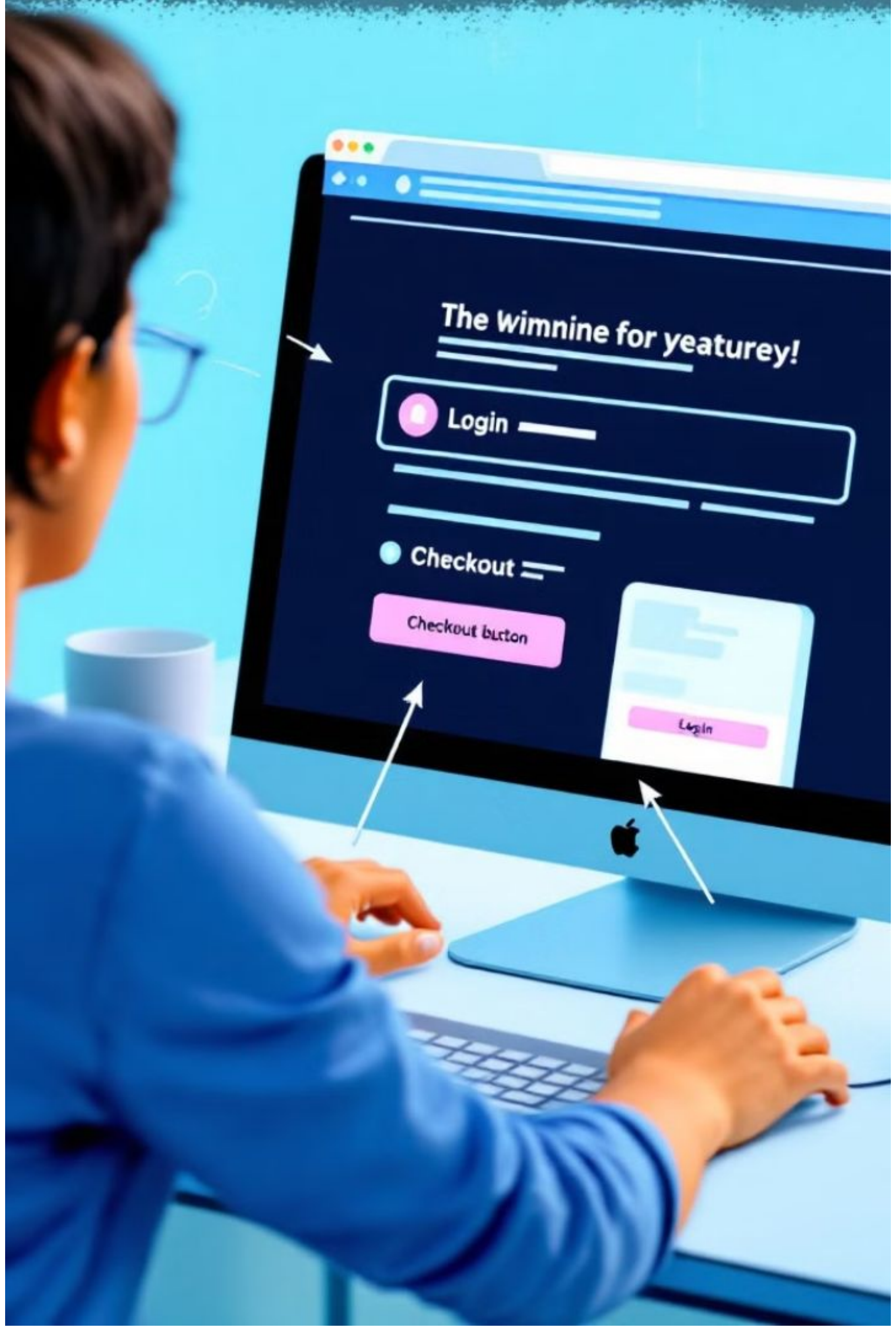
Core UML Diagram Types:

Structure Diagrams

Focus on the static components of a system, such as classes, objects, and their relationships. Examples include class diagrams, component diagrams, and deployment diagrams.

Behavior Diagrams

Illustrate the dynamic aspects of a system, such as interactions between objects, state transitions, and data flow. Examples include use case diagrams, sequence diagrams, and context diagrams.



Use Case Diagrams: Capturing System Reqs.

1. Represent the interactions between actors (users) and the system.
2. Capture functional requirements from a user perspective.
3. Help to define the scope and boundaries of a system.

Class Diagrams: Modeling Object Relationships

Classes

Represent data structures and behaviors of objects.

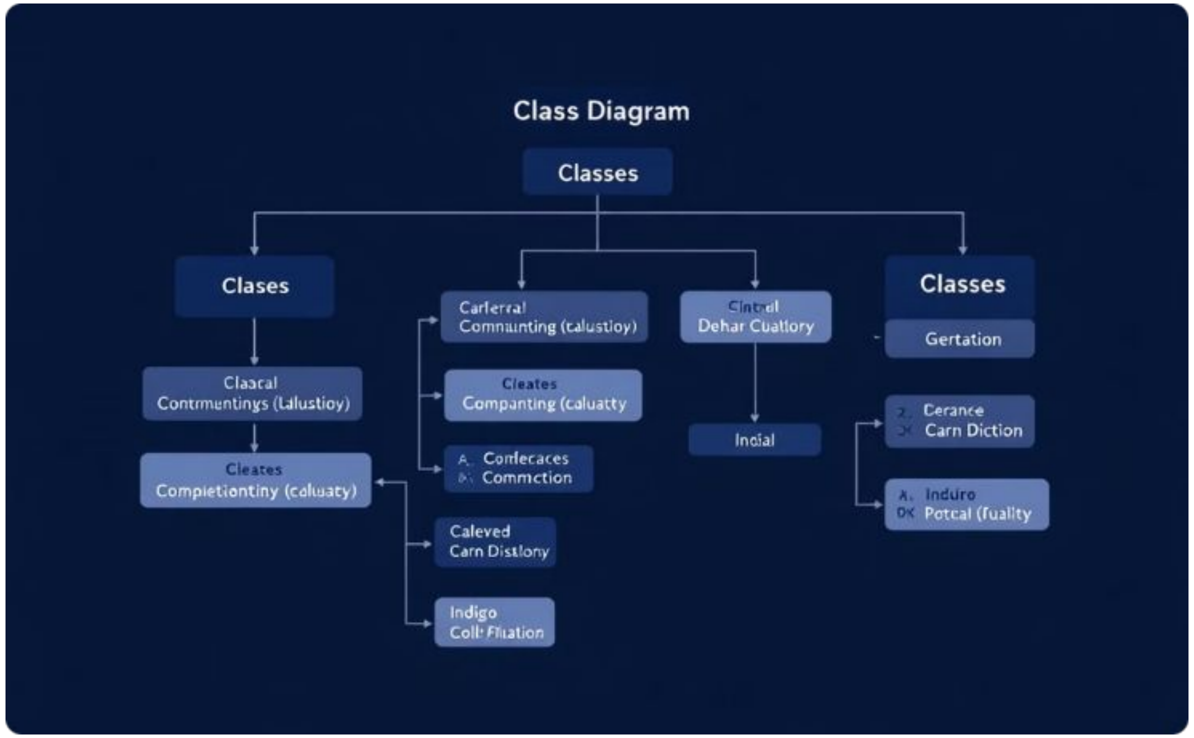
Relationships

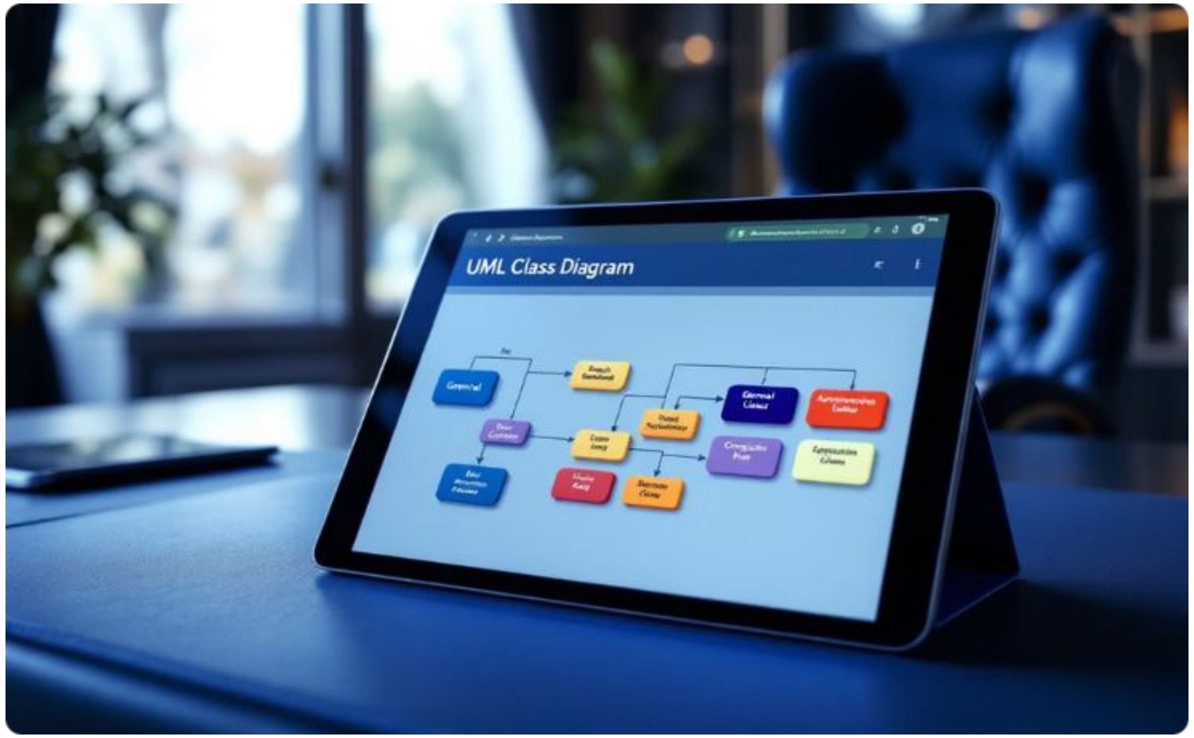
Show how classes interact, including inheritance, aggregation, and association.

Attributes and Methods

Define the properties and operations of classes.

Class Diagrams: Modeling Object Relationships





Classes and Attributes

Depict the structure and organization of a system by modeling classes as building blocks and their attributes as properties.

Relationships and Interactions

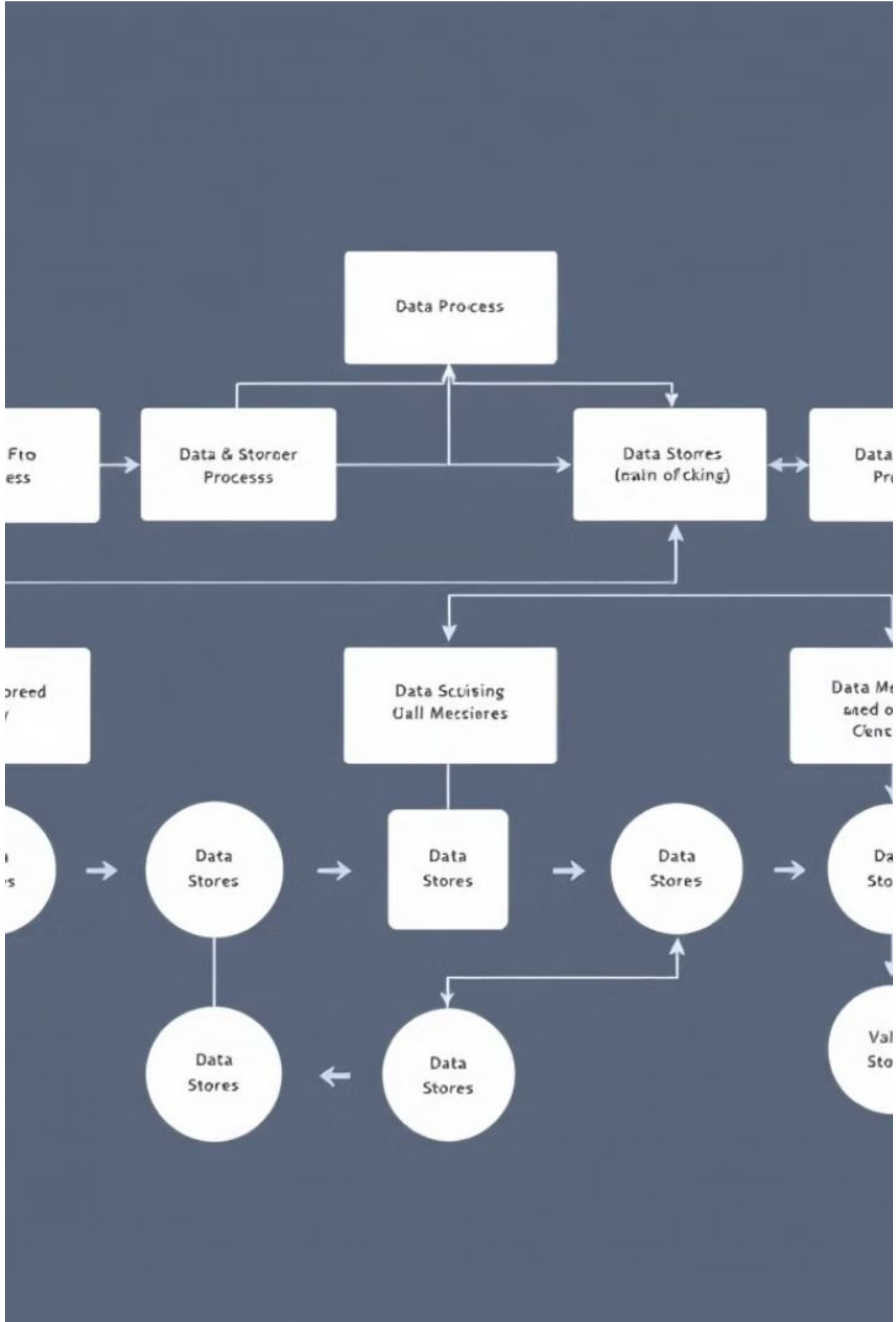
Represent relationships between classes, including inheritance (generalization), aggregation, and composition.

Data Flow Diagrams: Tracking Information Movement

1. Depict the movement of data through a system.

2. Identify processes, data sources, and destinations.

3. Help to understand how data is transformed and used within a system.

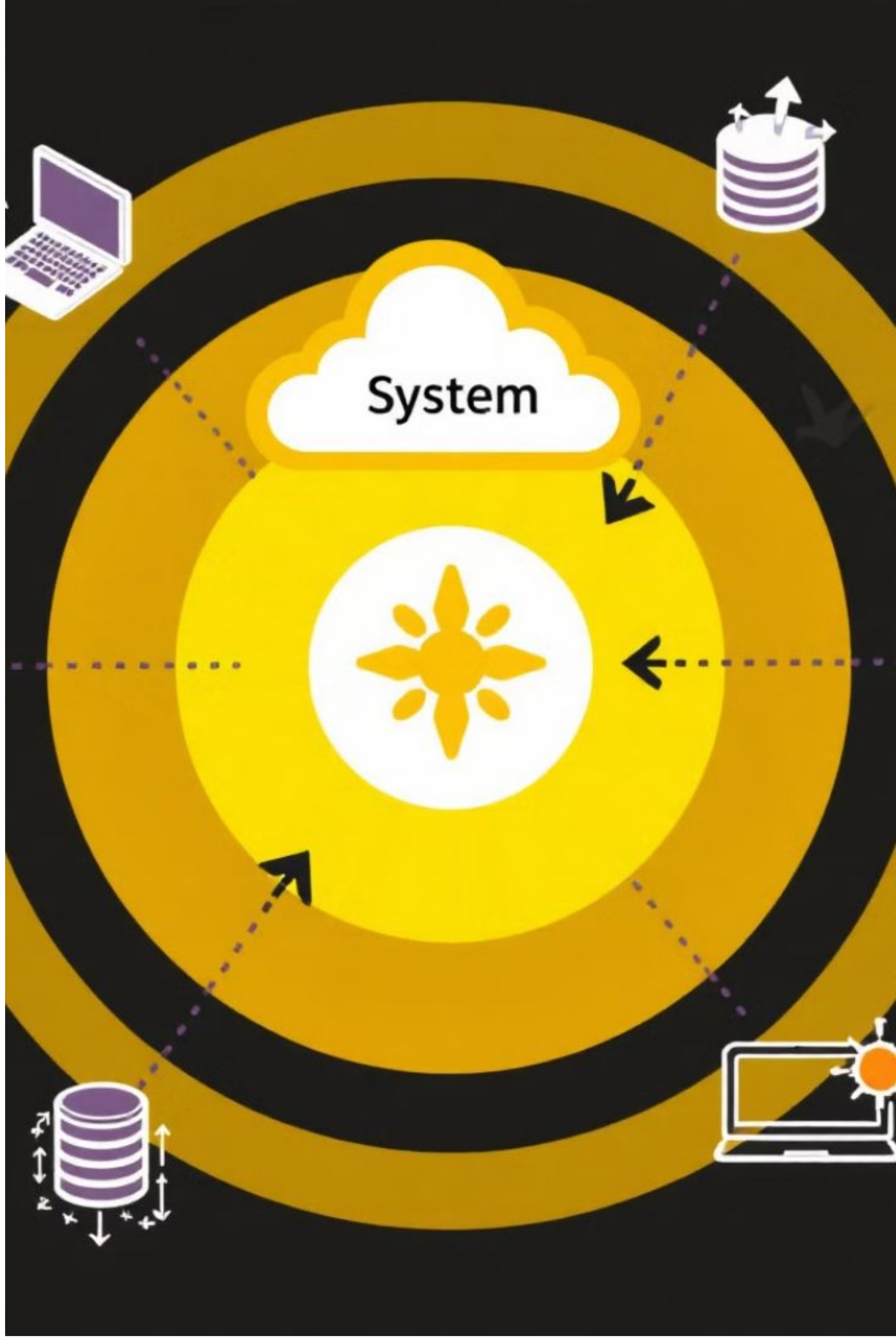


Context Diagrams: System Boundaries and Interactions

Define the scope of a system.

Show interactions with external entities.

Illustrate data flow across system boundaries.



Best Practices and Common Pitfalls to Avoid

- Keep diagrams simple and clear.
- Use consistent notation and symbols.
- Focus on communication and understanding.
- Avoid excessive detail and overly complex diagrams.