IARE
INSTITUTE OF
AERONAUTICAL ENGINEERING

Object Oriented Programming

Complex Systems

Dr. B. Surekha Reddy
IARE10795
Computer Science Engineering
Lecture Number – 2
Presentation Date – 16/09/2025

COURSE OBJECTIVES:

The students will try to learn:

I. The fundamental concepts and principles of object-oriented programming.

II. Advanced concepts of classes, and the need for constructors and destructors for developing well-structured and efficient programs.

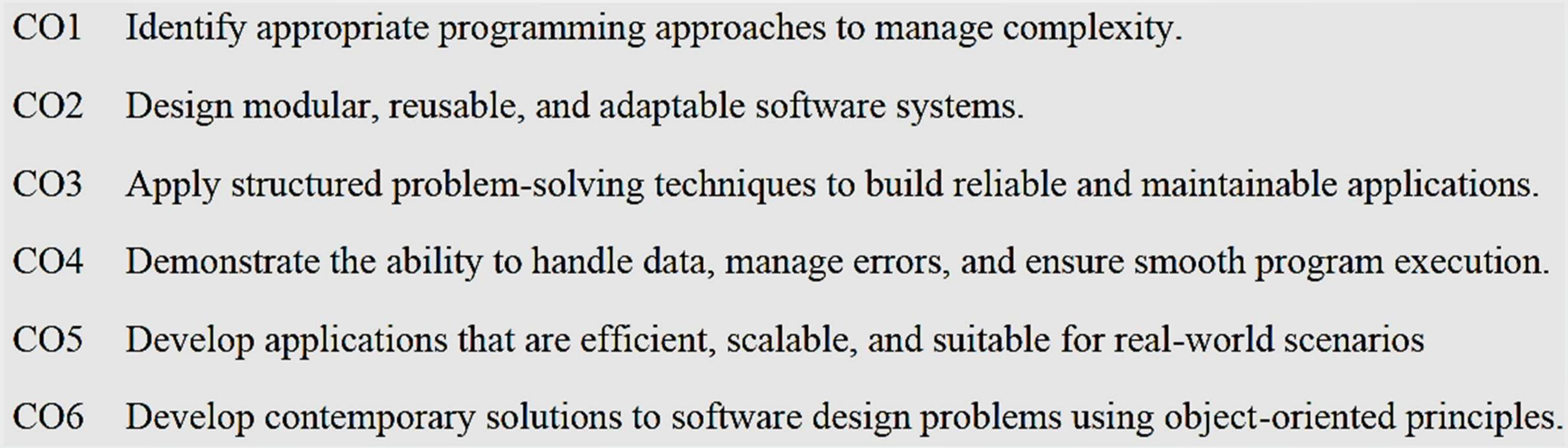
III. Overloaded casting operators when working on projects that involve complex data structures, numerical computations, or domain-specific operations.

IV. The implementation of the features such as inheritance, polymorphism, and encapsulation for tackling complex problems and creating well-organized, modular, and maintainable code.

V. The usage of I/O interfaces to transmit and receive data to solve real-time computing problems.

COURSE OUTCOMES:

After successful completion of the course, students should be able to:



CO1	Identify appropriate programming approaches to manage complexity.
CO2	Design modular, reusable, and adaptable software systems.
CO3	Apply structured problem-solving techniques to build reliable and maintainable applications.
CO4	Demonstrate the ability to handle data, manage errors, and ensure smooth program execution.
CO5	Develop applications that are efficient, scalable, and suitable for real-world scenarios
CO6	Develop contemporary solutions to software design problems using object-oriented principles.

Object Oriented Programming

MODULE - I: Object-oriented concepts (10)
Complex systems: definition, characteristics, and five attributes (hierarchy, abstraction, emergence, encapsulation, modularity).
Evolution of problem-solving: procedural vs. object-oriented thinking.
Objects as fundamental building blocks: state, behavior, and identity.
Benefits of OOP in managing complexity, Applications of OOP in real-world systems..

Definition of Complex Systems

- A complex system is a system composed of **multiple interconnected components** whose interactions give rise to behavior that is not easily predictable from the behavior of individual components.
- These systems often exhibit **nonlinear dynamics**, **adaptability**, and **emergent** properties, making them more than just the sum of their parts.
- **Examples** include ecosystems, the human brain, social networks, and the internet.

Characteristics of Complex Systems

- **Nonlinearity** – Small changes in one part of the system can produce disproportionately large effects elsewhere.
- **Interconnectedness** – Components interact in intricate ways, often with feedback loops.
- **Adaptability** – The system can change its behavior in response to environmental conditions.
- **Emergence** – System-level patterns or behaviors arise that are not present in individual components.
- **Dynamic Behavior** – Constantly evolving over time rather than remaining static.
- **Unpredictability** – Future states cannot be precisely forecasted due to the interactions among components.

Five Key Attributes of Complex Systems

1. Hierarchy
2. Abstraction
3. Emergence
4. Encapsulation
5. Modularity

Hierarchy, Abstraction, Encapsulation, Modularity → design principles we already know from OOP.

Emergence → added from *complex systems theory* to highlight that **sometimes large software systems show unexpected behaviors** (e.g., scalability issues, distributed interactions, AI behavior).

Five Key Attributes of Complex Systems

1. Hierarchy

- Complex systems are often structured in multiple layers or levels.
- Each layer can contain subsystems that interact to produce higher-level behavior.
- **Example**: Cells form tissues, tissues form organs, organs form organisms.

Object Oriented Programming

Abstraction –

- Abstraction is the concept of hiding the internal details and describing things in simple terms.
- **Hiding internal details** and showing functionality is known as **abstraction**.
- For example: phone call, we don’t know the internal processing.
- In java, we use abstract class and interface to achieve abstraction.

Object Oriented Programming

Abstraction –

- **Example:**
  - All are performing operations on the ATM machine like cash withdrawal, money transfer, retrieve mini-statement…etc. but we can’t know the internal details about ATM.

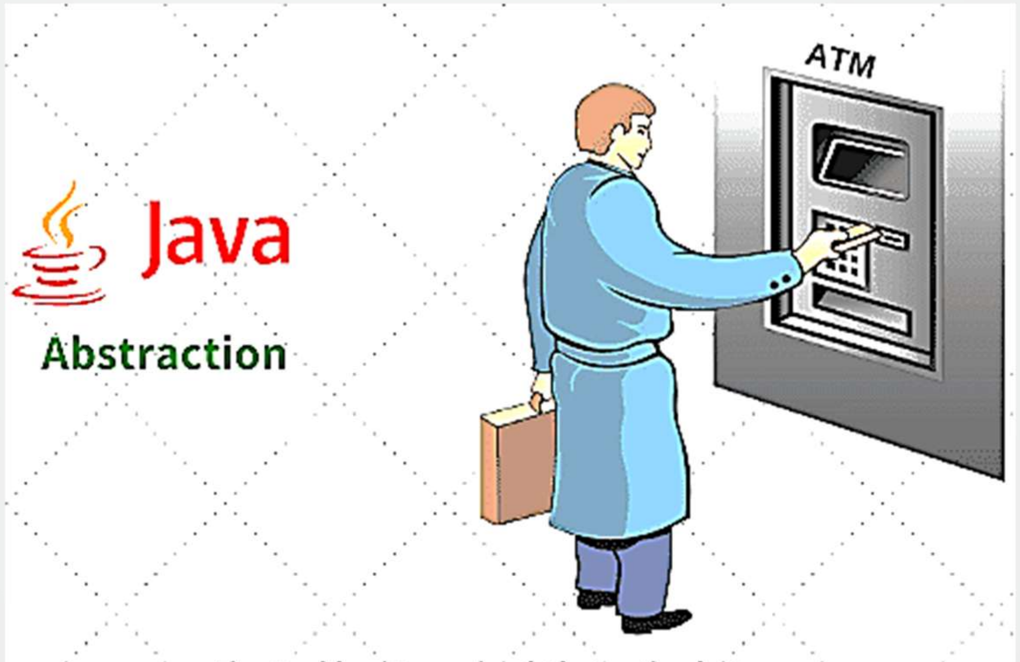


Fig: Realtime Example of Abstraction in Java

Five Key Attributes of Complex Systems

3. Emergence

- Emergence refers to patterns or behaviors that arise from interactions among system components but cannot be predicted by analyzing components individually.
- **Example**: Flocking behavior of birds, consciousness from neuronal interactions.

Emergence means:
When simple rules or interactions at a lower level give rise to unexpected or higher-level behaviors that cannot be fully explained by the lower-level parts alone.

Five Key Attributes of Complex Systems

4. Encapsulation

- Encapsulation hides the internal complexity of a component, exposing only what is necessary for interaction with other parts.
- Promotes modular design and reduces cognitive load when analyzing the system.
- **Example**: A software module provides a defined interface without revealing its internal logic.

Object Oriented Programming

Encapsulation –

- Encapsulation is the mechanism that **binds the data members of the class** with the **methods** together into a **single entity**.
- **Binding (or wrapping) code and data** together into a **single unit** is known as **encapsulation**.
- A java class is the example of encapsulation.
- A class typically consists of data members (data) and member functions (code that operate on the data) together.

Object Oriented Programming

Encapsulation –

- Example:
  - Capsule, it is wrapped with different medicines. In a capsule, all medicine is encapsulated inside a capsule.

- A Java class is an example of encapsulation.
- Java bean is the fully encapsulated class because all the data members are private here.

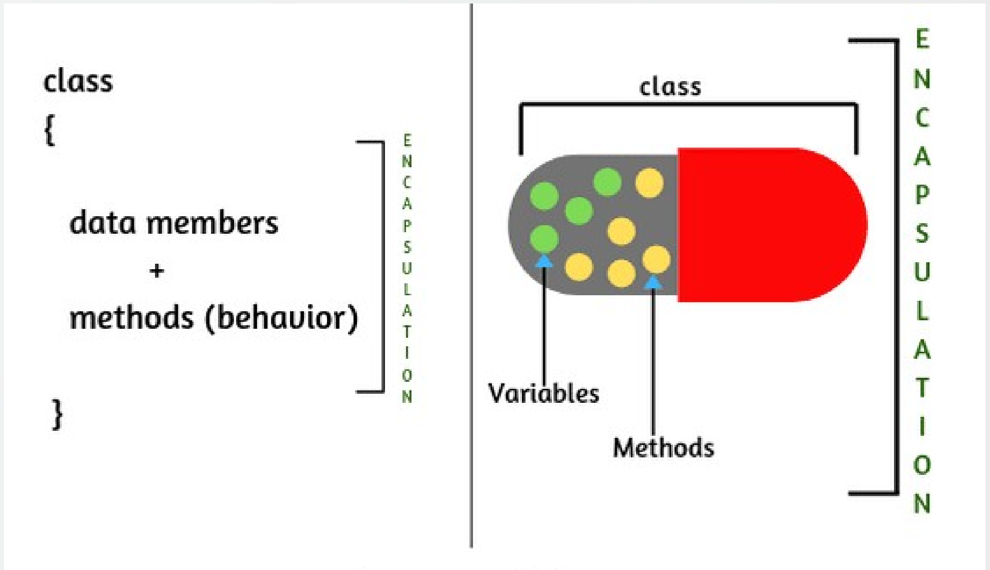


Fig: Encapsulation

Five Key Attributes of Complex Systems

5. Modularity

- Modularity divides the system into discrete, functionally independent units or modules.
- Improves robustness, adaptability, and ease of maintenance.
- **Example**: A car engine composed of separate modules like fuel injection, cooling, and transmission.