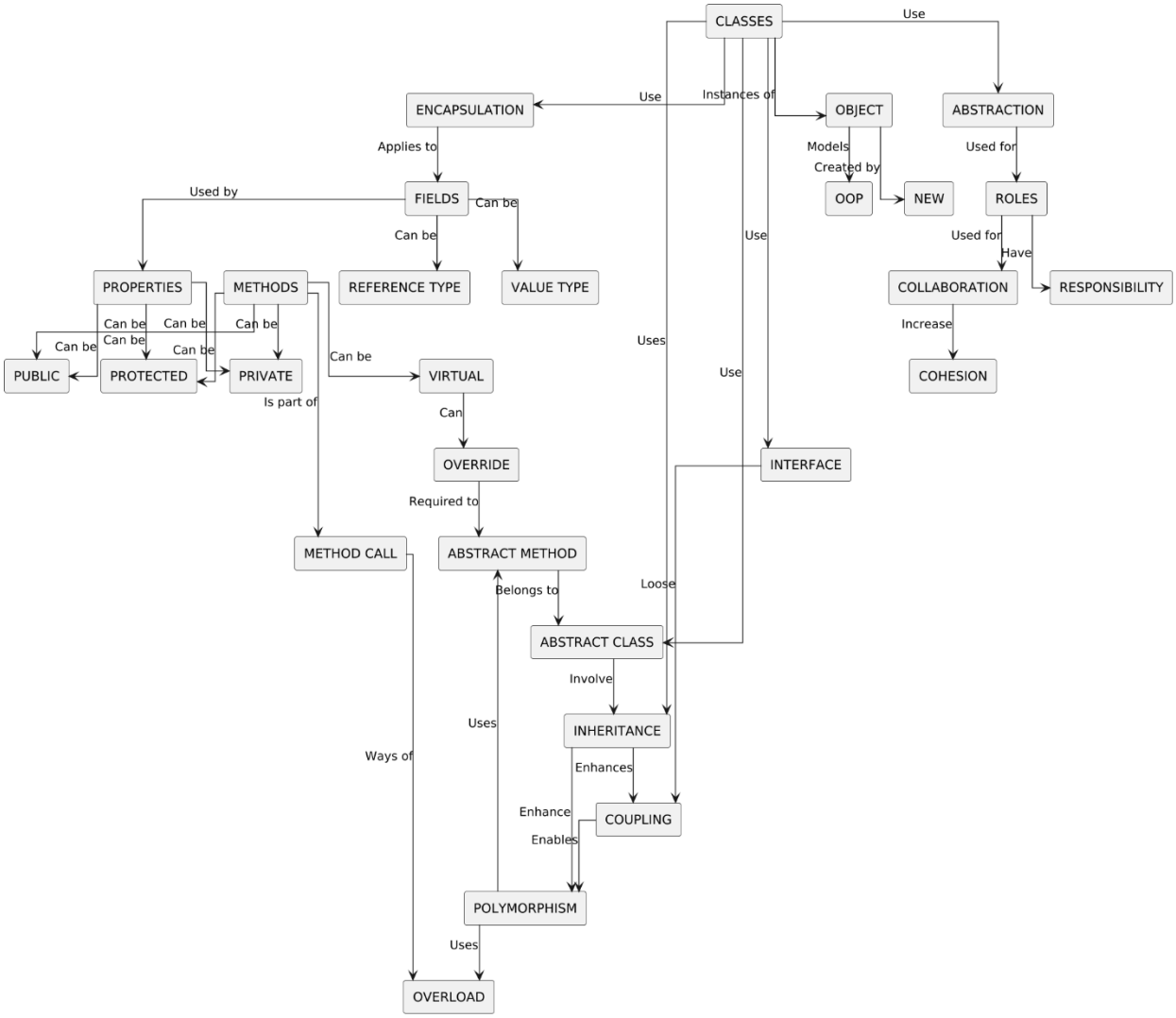

1. Abstraction

- Abstraction in object-oriented programming (OOP) simplifies complex systems by emphasizing essential roles and responsibilities while hiding irrelevant details.
- It allows developers to model real-world objects or processes using only the necessary data and behaviors.
- This principle helps reduce complexity, improves modularity, and offers a cleaner, more focused interface for interacting with objects.
- A common application of abstraction is in UML diagrams—within a class diagram, we mainly see the roles and responsibilities of attributes and methods, without needing to understand the implementation. This abstraction makes the system easier to understand and use.

---

2. Inheritance

- Inheritance is a core concept of OOP where a class (child or subclass) can inherit the attributes and behaviors of another class (parent or superclass).
- It promotes code reuse, flexibility, and scalability by establishing an "is-a" relationship—for example, a Player class could inherit from a more general GameObject class.
- This allows shared functionality to be written once and reused across multiple subclasses.

---

3. Encapsulation

- Encapsulation involves bundling data and the methods that operate on that data into a single unit, typically a class.
- It protects internal object states by restricting direct access—data is kept private, and interaction happens through public methods.
- This “black box” approach ensures data integrity, enhances modular design, and promotes reuse.
- A good example is the Clock task, where internal variables like _hours, _minutes, and _seconds are shielded from direct access and managed through defined methods.

---

4. Polymorphism

- Polymorphism means “many forms” and allows objects of different types to be treated through a common interface.
- It supports both compile-time (method overloading) and runtime (method overriding) flexibility.
- This principle boosts code reusability and adaptability by letting developers write generalized code that works across various object types.
- A practical example is a draw() method that can be used to render different shapes like triangles, circles, or lines, depending on the specific object passed in.