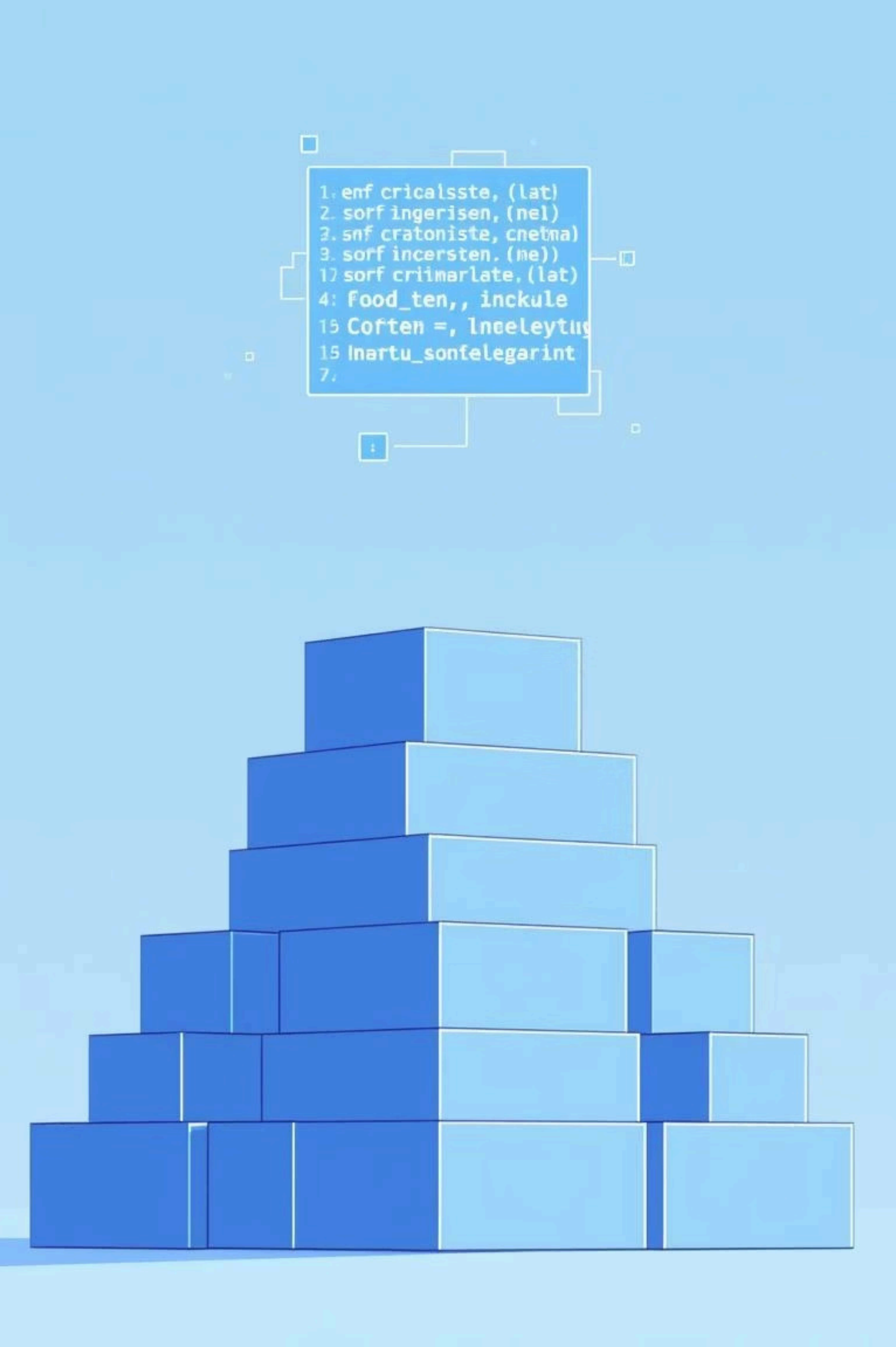


The Four Pillars of Object-Oriented Programming

Class Agenda

- Four pillars of OOP
- Mini-project in OOP x 2

Pillar 1: Abstraction

Definition

Abstraction hides complex implementation details.

Real-World Example

A car's "Start" button abstracts away the complex process of engine ignition.

Only show what is needed, hide the rest.

Abstraction Example: Coffee Machine

User Experience

User simply presses "Make Coffee" button and receives coffee.

Hidden Complexity

Internal processes like water heating, bean grinding, brewing pressure, and temperature control are all abstracted away.

Code equivalent: A Coffee class with a simple `brew()` method that handles all the complex operations internally.

Pillar 2: Encapsulation

Definition

Encapsulation bundles related data (attributes) and methods (behaviors) into a single unit (class) and restricts direct access to some of its components.

Key Features

Uses access modifiers (`private`, `protected`, `public`) to control visibility and access to class members.

Benefits

Protects **data integrity**, prevents unintended modifications.

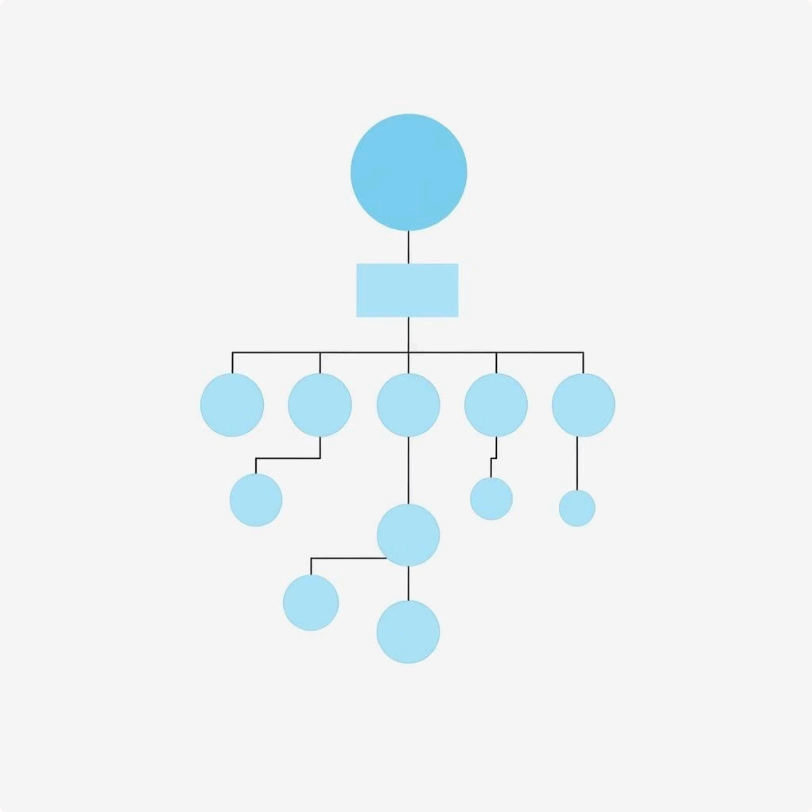
Encapsulation Example: Bank Account

```
class BankAccount {
  private double balance;  // Private data

  public double getBalance() {
    return balance;  // Controlled access
  }

  public void deposit(double amount) {
    if (amount > 0) {
      balance += amount;
    }
  }

  public boolean withdraw(double amount) {
    if (amount <= balance && amount > 0) {
      balance -= amount;
      return true;
    }
    return false;
  }
}
```

Encapsulation Benefits Demonstrated

- Balance variable is **private** – cannot be directly accessed or modified
- Deposit method ensures only **positive amounts** can be added
- Withdraw method prevents **overdrafts** by checking available balance
- Data integrity is preserved through **controlled access** points

Pillar 3: Inheritance



Definition

Inheritance allows a class **(child/subclass)** to inherit properties and behaviors from another class **(parent/superclass).**

Key Benefits

- Promotes **code reuse** by inheriting existing functionality
- Creates **logical hierarchies** of related classes
- Enables **specialization** through extending parent classes
- Reduces redundancy and maintenance overhead

Example: A `Car` class inherits from a `Vehicle` class, gaining common features like `start()` and `stop()` methods.

Made with GAMMA

Inheritance Example: Animals

Animal Class (Parent)

```
class Animal {
  void eat() { ... }
  void sleep() { ... }
}
```

Dog Class (Child)

```
class Dog extends Animal {
  void bark() { ... }
  // Inherits eat() and sleep()
}
```

Cat Class (Child)

```
class Cat extends Animal {
  void meow() { ... }
  // Inherits eat() and sleep()
}
```

Common behaviors are **defined once** in the parent class and reused across all child classes.

Pillar 4: Polymorphism

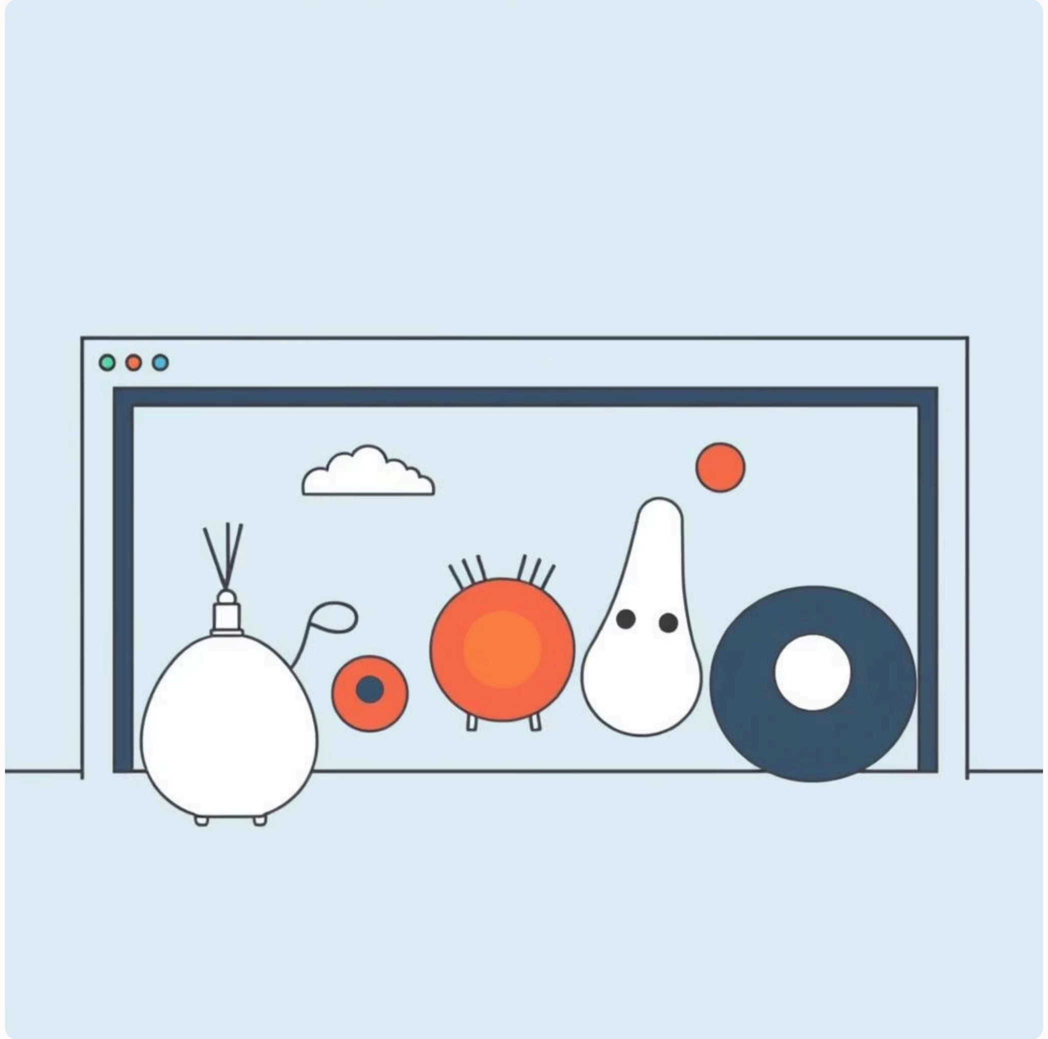
Definition

Polymorphism allows objects of different classes to be treated as instances of a **common parent class,** with **methods behaving differently** based on the actual object type.

Types of Polymorphism

- **Compile-time** (method overloading)
- **Runtime** (method overriding)

```
// Different objects, same method call
Animal dog = new Dog();
Animal cat = new Cat();
dog.makeSound();  // Outputs: "Woof!"
cat.makeSound();  // Outputs: "Meow!"
```



This enables flexible, extensible code that can **handle new derived classes without changing existing code.**

Why OOP Pillars Matter

1

Abstraction

Simplifies complexity by hiding implementation details behind clean interfaces

2

Encapsulation

Protects data integrity by controlling access and modification

3

Inheritance

Promotes code reuse and establishes logical hierarchies

4

Polymorphism

Enables flexible behavior through a common interface