Общие вопросы:

1. Что такое ООП? – полное определение

Объектно ориентированное программирование — это парадигма программирования, основанная на концепции объектов, которые объединяют данные (поля, атрибуты) и методы (функции).

Объединяет следующие понятия:

Инкапсуляция: Сокрытие внутренней реализации объекта и предоставление доступа только через публичные методы.

Пример: Класс BankAccount, где баланс изменяется только через методы deposit() и withdraw().

Наследование: Создание новых классов на основе существующих с возможностью переопределения или расширения функциональности.

Пример: Класс Animal → подклассы Dog, Cat.

Полиморфизм: Возможность использования объектов разных классов через единый интерфейс.

Пример: Метод draw() для классов Circle и Square в графическом редакторе.

А бстракция: Упрощение сложных систем путем выделения ключевых характеристик и игнорирования несущественных деталей.

Пример: Интерфейс Vehicle с методами startEngine(), move().

2. Магическое число 7 Миллера – примеры из IT

Психолог Джордж Миллер установил, что человек может одновременно удерживать в рабочей памяти 7±2 элемента. В IT это проявляется в:

Лимит вложенности циклов/условий (рекомендуется не более 3–5, но 7 — критический порог).

Количество параметров функции (например, в API лучше не превышать 7 аргументов).

Меню интерфейса (7 основных пунктов — максимум для удобства пользователя).

Количество классов в модуле (в Python, например, sys.maxsize не случайно имеет порядок ~2³²).

Длина строки кода (80–120 символов — косвенно связано с восприятием).

Число уровней вложенности папок (глубина >7 усложняет навигацию).

Количество элементов в CLI-команде (например, git log --oneline -n 7).

3. Энтропия ПО – негэнтропийные меры

Энтропия ПО — мера хаоса в коде (рост сложности, беспорядка). Негэнтропийные меры (уменьшающие энтропию):

Рефакторинг (разбиение монолита на микросервисы).

Документация (например, Swagger для API).

Стандарты кодирования (PEP 8, ESLint).

Модульность (использование паттернов like MVC).

Автоматизированное тестирование (юнит-тесты снижают хаос изменений).

4. 5 признаков сложной системы по Гради Бучу

Иерархичность

Пример 1: В лабораторной по ООП — класс Car → Engine → Piston.

Пример 2: В pet-проекте (веб-приложение) App → Router → Controller → Model.

Наличие обратных связей

Пример 1: Логика обработки ошибок в API (возврат HTTP 400 → коррекция входных данных).

Пример 2: Автотесты, которые влияют на доработку кода.

Эмерджентность (свойства системы ≠ сумме свойств частей)

Пример 1: Баг в многопоточном приложении, которого нет в отдельных потоках.

Пример 2: Неожиданное поведение микросервисов при масштабировании.

Динамическая изменчивость

Пример 1: Изменение состояния объекта User от anonymous к authenticated.

Пример 2: Адаптация UI под разные разрешения экрана.

Децентрализованное управление

Пример 1: Git-репозиторий с ветками main/feature.

Пример 2: БД с шардингом (разные узлы обрабатывают запросы).

5. Закон иерархических компенсаций Седова – примеры в IT

Закон: Рост разнообразия на верхнем уровне системы требует уменьшения разнообразия на нижних уровнях.

Унификация процессоров (x86, ARM) → разнообразие софта (ОС, приложения).

Стандарты HTTP/IP → миллионы веб-сайтов.

Языки высокого уровня (Python, JS) → сокращение ассемблерного кода.

Облачные платформы (AWS, Azure) → упрощение развертывания инфраструктуры.

Фреймворки (React, Django) → меньше "велосипедов" в разработке.

Каждый пример показывает, как стандартизация "низов" позволяет развивать "верхи".