**1. Порождающие паттерны проектирования**

Определение:

Порождающие паттерны — это шаблоны проектирования, которые решают задачи создания объектов, обеспечивая гибкость, повторное использование кода и контроль над процессом инстанцирования.

Основные паттерны:

1. Singleton (Одиночка)

- Назначение: Гарантирует, что у класса есть только один экземпляр, и предоставляет глобальную точку доступа к нему.

- Пример использования: Логгеры, подключения к базе данных.

2. Factory Method (Фабричный метод)

- Назначение: Определяет интерфейс для создания объекта, но оставляет подклассам решение о том, какой класс инстанцировать.

- Пример: Создание документов в текстовом редакторе (разные форматы: PDF, DOCX).

3. Abstract Factory (Абстрактная фабрика)

- Назначение: Создаёт семейства связанных объектов без указания их конкретных классов.

- Пример: GUI-библиотеки (кнопки, окна для Windows/macOS).

4. Builder (Строитель)

- Назначение: Позволяет создавать сложные объекты пошагово, отделяя конструирование от представления.

- Пример: Конструирование SQL-запросов.

5. Prototype (Прототип)

- Назначение: Копирует существующие объекты без зависимости от их классов.

- Пример: Клонирование игровых персонажей.

Преимущества:

- Уменьшение дублирования кода.

- Гибкость в создании объектов.

**2. Оценка сложности алгоритма**

Определение:

Оценка сложности алгоритма — это анализ времени выполнения (временная сложность) и использования памяти (пространственная сложность) в зависимости от размера входных данных.

Big O нотация:

- O(1) — Константная сложность (доступ к элементу массива).

- O(n) — Линейная сложность (поиск в неотсортированном массиве).

- O(n²) — Квадратичная сложность (пузырьковая сортировка).

- O(log n) — Логарифмическая сложность (бинарный поиск).

- O(n log n) — Сортировка слиянием.

Практическое применение:

- Выбор оптимального алгоритма для больших данных.

- Сравнение производительности решений.

**3. Режимы адресации в системном ПО**

Определение:

Режимы адресации определяют, как процессор вычисляет адрес операнда в памяти или регистре.

Основные режимы:

1. Прямая адресация

- Адрес указывает непосредственно на ячейку памяти.

- Пример: `MOV AX, [1234h]` (загрузить значение из адреса 1234h).

2. Косвенная адресация

- Адрес хранится в регистре.

- Пример: `MOV AX, [BX]` (загрузить значение по адресу из BX).

3. Индексная адресация

- Адрес = базовый регистр + смещение.

- Пример: `MOV AX, [SI+10h]`.

4. Относительная адресация

- Адрес вычисляется относительно счётчика команд (PC).

- Используется в командах перехода (JMP).

Применение:

- Оптимизация доступа к памяти.

- Поддержка сложных структур данных (массивы, указатели).

**4. Виды ошибок в программировании**

Классификация:

1. Синтаксические ошибки

- Нарушение правил языка (отсутствие точки с запятой, скобок).

- Обнаруживаются компилятором.

2. Логические ошибки

- Программа работает, но выдаёт неверный результат.

- Пример: Неправильная формула в расчётах.

3. Ошибки времени выполнения (Runtime errors)

- Возникают при выполнении (деление на ноль, доступ к null-указателю).

4. Ресурсные ошибки

- Утечки памяти, исчерпание стека.

Стратегии обработки:

- Использование исключений (try-catch).

- Валидация входных данных.

- Логирование ошибок.

**5. Паттерны поведения**

Определение:

Паттерны поведения определяют способы взаимодействия объектов и распределения ответственности.

Примеры:

1. Observer (Наблюдатель)

- Объекты подписываются на события и уведомляются об изменениях.

- Пример: Уведомления в GUI.

2. Strategy (Стратегия)

- Инкапсулирует алгоритмы в отдельные классы, позволяя выбирать их на лету.

- Пример: Сортировка (QuickSort, BubbleSort).

3. Command (Команда)

- Инкапсулирует запрос как объект, позволяя ставить операции в очередь.

- Пример: Кнопки в интерфейсе.

Преимущества:

- Уменьшение связанности кода.

- Гибкость в изменении поведения системы.

**6. Уровни тестирования ПО**

Основные уровни:

1. Модульное тестирование (Unit Testing)

- Проверка отдельных функций/методов.

- Инструменты: JUnit (Java), pytest (Python).

2. Интеграционное тестирование

- Проверка взаимодействия модулей.

- Пример: Тестирование API.

3. Системное тестирование

- Проверка всей системы на соответствие требованиям.

4. Приёмочное тестирование (UAT)

- Проверка заказчиком перед сдачей.

Цели:

- Раннее обнаружение ошибок.

- Обеспечение качества кода.

**7. Декораторы в программировании**

Определение:

Декоратор — паттерн, который динамически добавляет объекту новые обязанности, не изменяя его класс.

Применение:

- Логирование.

- Кеширование.

- Проверка прав доступа.

**8. Структура Android-приложения**

Основные компоненты:

1. AndroidManifest.xml

- Описание приложения (разрешения, активности).

2. Activities

- Экраны приложения (жизненный цикл: onCreate, onStart).

3. Resources (res/)

- Макеты (layout), строки (strings), изображения (drawable).

4. Gradle

- Система сборки (зависимости, версии SDK).

**9. Архитектура системных программ**

Компоненты:

1. Ядро ОС

- Управление процессами, памятью, устройствами.

2. Драйверы устройств

- Обеспечивают взаимодействие с железом.

3. Системные утилиты

- Форматирование дисков, настройка сети.

Примеры:

- Монолитные ядра (Linux).

- Микроядра (QNX).

**10. Стратегии проектирования тестовых данных**

Методы:

1. Эквивалентное разделение

- Разбиение входных данных на классы (например, положительные/отрицательные числа).

2. Граничные значения

- Тестирование на границах диапазонов (min, max).

3. Случайные данные

- Fuzz-тестирование для проверки устойчивости.

**11. ООП: Наследование**

Определение:

Наследование позволяет классу (потомку) перенимать свойства и методы другого класса (родителя).

Преимущества:

- Повторное использование кода.

- Иерархическая организация классов.

Недостатки:

- Риск создания сложных иерархий.

**12. Ошибки: виды и стратегии работы**

Типы ошибок:

- Ошибка (Error) — действие разработчика.

- Дефект (Defect) — результат ошибки в коде.

- Сбой (Failure) — отказ системы при выполнении.

Стратегии:

1. Предотвращение:

- Code reviews, статический анализ.

2. Обнаружение:

- Тестирование, логирование.

3. Обработка:

- Исключения (try-catch), восстановление состояния.

**13. Автоматизация технической документации**

Инструменты:

1. Doxygen

- Генерация документации из комментариев (C++, Java).

2. Sphinx

- Документирование Python-проектов (используется в Django).

3. Swagger

- Документирование REST API (OpenAPI).

Преимущества:

- Актуальность документации.

- Снижение трудозатрат.

**14. Обмен данными через именованные каналы (FIFO)**

Определение:

Именованные каналы — это файлы особого типа, позволяющие процессам обмениваться данными.

Применение:

- Межпроцессное взаимодействие (IPC).

- Обмен данными между независимыми программами.

**15. Жизненный цикл активности Android**

Этапы:

1. onCreate() — создание активности (инициализация UI).

2. onStart() — активность становится видимой.

3. onResume() — активность получает фокус.

4. onPause() — частичное перекрытие (например, диалог).

5. onStop() — активность больше не видна.

6. onDestroy() — уничтожение активности.

Важность:

- Корректное управление ресурсами (память, данные).

**16. Ошибка, дефект, сбой: различия**

- Ошибка (Mistake) — человеческое действие (опечатка в коде).

- Дефект (Defect/Bug) — следствие ошибки в программном коде.

- Сбой (Failure) — отклонение работы программы от требований.

Пример:

- Ошибка: Программист забыл проверить деление на ноль.

- Дефект: Код `x = 10 / 0`.

- Сбой: Программа аварийно завершается при выполнении.

**17. ConstraintLayout в Android**

Определение:

ConstraintLayout — гибкий контейнер для компоновки UI-элементов с помощью ограничений (constraints).

Преимущества:

- Отсутствие вложенности (в отличие от LinearLayout).

- Адаптивность под разные экраны.

Применение:

- Сложные макеты с динамическим позиционированием.

**18. Мьютексы для синхронизации**

Определение:

Мьютекс (mutex) — примитив синхронизации, который предотвращает одновременный доступ к общему ресурсу.

Проблемы:

- Взаимная блокировка (deadlock) — если поток не освобождает мьютекс.

- Голодание (starvation) — если поток долго ждёт мьютекс.

Альтернативы:

- Семафоры, атомарные операции.

**19. Алгоритмы на графах**

Основные алгоритмы:

1. Обход в глубину (DFS)

- Рекурсивный обход всех вершин.

- Применение: Поиск циклов, топологическая сортировка.

2. Обход в ширину (BFS)

- Поиск кратчайшего пути в невзвешенном графе.

3. Алгоритм Дейкстры

- Поиск кратчайшего пути во взвешенном графе.

Применение:

- Маршрутизация в сетях.

**20. Парадигма "Разделяй и властвуй"**

Определение:

Алгоритм разбивает задачу на подзадачи, решает их рекурсивно и объединяет результаты.

Примеры:

- Быстрая сортировка (QuickSort) .

- Сортировка слиянием (MergeSort) .

Алгоритм QuickSort:

1. Выбрать опорный элемент (pivot).

2. Разделить массив на две части: элементы меньше pivot и больше.

3. Рекурсивно отсортировать подмассивы.

Преимущества:

- Эффективность (O(n log n) в среднем).

**21. Дескрипторы и псевдодескрипторы**

Дескрипторы:

- Это структуры данных, описывающие ресурсы (файлы, сокеты).

- Пример: Дескриптор файла в Linux (`int fd = open("file.txt")`).

Псевдодескрипторы:

- Абстракции, не связанные с реальными ресурсами.

- Пример: `stdin`, `stdout` в C.

Применение:

- Управление ресурсами ОС.

**22. ООП: Инкапсуляция**

Определение:

Инкапсуляция — это скрытие внутренней реализации класса и предоставление контролируемого доступа к данным.

Преимущества:

- Защита данных от некорректного доступа.

- Упрощение модификации кода.

**23. ООП: Классы и объекты**

Класс:

- Шаблон для создания объектов (определяет поля и методы).

Объект:

- Экземпляр класса с конкретными значениями полей.

Применение:

- Моделирование сущностей предметной области.

**24. Методы отладки**

Основные методы:

1. Логирование — запись событий в файл.

2. Точки останова (breakpoints) — пауза выполнения для инспекции.

3. Пошаговое выполнение — выполнение кода строка за строкой.

Инструменты:

- IDE: PyCharm, Visual Studio.

- Консольные: GDB (C/C++), pdb (Python).

**25. Процессы и взаимодействие между ними**

Процесс:

- Экземпляр выполняемой программы с собственным адресным пространством.

Способы взаимодействия (IPC):

1. Каналы (Pipes) — однонаправленный обмен данными.

2. Очереди сообщений — структурированный обмен.

3. Разделяемая память — общий участок памяти.

Проблемы:

- Синхронизация доступа к ресурсам.

**26. Структура консольного приложения**

Основные компоненты:

1. Точка входа: Функция `main()`.

2. Логика: Обработка аргументов командной строки.

3. Ввод-вывод: Чтение из `stdin`, запись в `stdout`.

Этапы разработки:

1. Обработка аргументов.

2. Выполнение основной логики.

3. Вывод результатов.

**27. Контейнеры компоновки макетов: LinearLayout**

LinearLayout — это контейнер в Android, который размещает дочерние элементы по одной линии — вертикально или горизонтально. Он позволяет управлять выравниванием, отступами и весом (layout\_weight) элементов для гибкой верстки. Например, вертикальный LinearLayout используется для размещения форм, полей ввода и кнопок сверху вниз.

**Вопрос 28. Организация сетевого взаимодействия**

Сетевое взаимодействие в ПО реализуется с помощью протоколов (например, HTTP, TCP/IP), библиотек (например, Retrofit, OkHttp, Volley) и API. Ключевые этапы: установление соединения, формирование запроса, обработка ответа, обработка ошибок. При этом важны безопасность (HTTPS, OAuth), обработка JSON/XML, асинхронность и обработка таймаутов.

**Вопрос 29. Событийно-управляемое программирование**

Это модель, в которой выполнение программы зависит от внешних и внутренних событий: пользовательских действий, получения данных, таймеров и т. д. Программа содержит обработчики событий (event handlers), которые срабатывают при наступлении события. Эта модель активно используется в GUI-приложениях и веб-разработке.

**Вопрос 30. Система информационной безопасности системного ПО**

Включает методы и средства защиты ОС и системных компонентов: контроль доступа (ACL), аутентификацию пользователей, шифрование, защиту от вирусов, аудит событий, патч-менеджмент, предотвращение переполнения буфера. Также важна изоляция процессов и защита памяти.

**Вопрос 31. Коллекции**

Коллекции — это структуры данных для хранения групп объектов. В Java есть интерфейсы List, Set, Map и их реализации (ArrayList, HashSet, HashMap). Коллекции позволяют добавлять, удалять, сортировать и искать элементы, часто с использованием итераторов и лямбда-выражений.

**Вопрос 32. Алгоритмы поиска**

Примеры: линейный поиск (поиск по каждому элементу), бинарный поиск (на отсортированных данных), поиск в графах (DFS, BFS), хеш-поиск (использует хеш-функцию). Выбор зависит от структуры данных и задачи.

**Вопрос 33. Верификация программного обеспечения**

Процесс подтверждения того, что программное обеспечение соответствует техническим требованиям. Используются статический анализ, инспекция кода, формальные методы, доказательства корректности. Отличается от валидации, которая проверяет соответствие ожиданиям пользователя.

**Вопрос 34. Методы тестирования**

* **Модульное** — проверка отдельных компонентов
* **Интеграционное** — взаимодействие компонентов
* **Системное** — проверка всей системы
* **Приемочное** — проверка заказчиком По доступу к коду: черный ящик (без знания кода), белый ящик (с доступом к коду), серый ящик (частичный доступ).

**Вопрос 35. Методы оптимизации программного кода**

Используются методы профилирования, устранение дублирующего кода, эффективные алгоритмы и структуры данных, lazy loading, кеширование, упрощение логики. Также важно избегать утечек памяти и учитывать многопоточность.

**Вопрос 36. Порядок разработки тестов. Аксиомы тестирования**

Тесты разрабатываются на этапе проектирования (TDD, BDD). Аксиомы: тестирование выявляет ошибки, полное тестирование невозможно, раннее тестирование снижает издержки, бессмысленно повторять одни и те же тесты без изменений.

**Вопрос 37. Классификация типов мобильных приложений по способу разработки**

* **Нативные** — пишутся на языке платформы (Java/Kotlin для Android, Swift/Obj-C для iOS), высокая производительность.
* **Кроссплатформенные** — один код (Flutter, React Native), ниже производительность, выше экономия.
* **Веб-приложения** — работают в браузере, не требуют установки. Плюсы/минусы зависят от бюджета, требований и платформ поддержки.

**Вопрос 38. Примитивы синхронизации: События**

События позволяют одному потоку сигнализировать другим о наступлении определенного состояния. Используются, чтобы синхронизировать начало выполнения после подготовки ресурсов или завершения другого потока. Пример — Event в Python или CountDownLatch в Java.

**Вопрос 39. Принципы разработки интерфейса пользователя**

Главные принципы: простота, интуитивность, последовательность, адаптивность, отзывчивость, доступность. Применяются гайдлайны платформ (Material Design, Human Interface Guidelines). Пользователь должен быстро ориентироваться в интерфейсе.

**Вопрос 40. Определение потока. Создание и завершение потока**

Поток — единица выполнения, параллельная другим потокам. В Java создается через наследование Thread или реализацию Runnable. Запускается методом start(), завершение — по окончании run() или принудительно (не рекомендуется).

**Вопрос 41. Контейнеры компоновки макетов: FrameLayout**

FrameLayout — простой контейнер Android, размещает все дочерние элементы в верхнем левом углу, элементы могут перекрываться. Подходит для отображения одного элемента или наложения (например, кнопки поверх видео).

**Вопрос 42. Место тестирования и отладки в жизненном цикле программного обеспечения**

После этапа реализации идет тестирование, где выявляются и исправляются ошибки. Отладка используется во всех этапах. Повторяется при каждом изменении, особенно важно при рефакторинге или добавлении нового функционала.

**Вопрос 43. Технология структурного программирования**

Методология разработки с использованием последовательности, ветвления и циклов. Исключает использование goto. Повышает читаемость, облегчает анализ и сопровождение кода. Основа для большинства языков высокого уровня.

**Вопрос 44. Состояния потока**

Основные состояния: New, Runnable, Blocked, Waiting, Timed Waiting, Terminated. Переходы зависят от системных вызовов (start(), sleep(), wait(), notify(), join() и др.).

**Вопрос 45. Алгоритмы сортировки**

Сравнительные: пузырьковая, вставками, выбором. Эффективные: быстрая, слиянием, пирамидальная. Встроенные сортировки используют адаптивные методы. Сортировка может быть стабильной (сохраняет порядок равных элементов).

**Вопрос 46. Системное программное обеспечение: подходы к определению и классификации**

Это ПО, управляющее аппаратной частью: ОС, драйверы, утилиты, трансляторы. Классифицируется по функциям: управления, защиты, обеспечения среды выполнения.

**Вопрос 47. Использование интентов при разработке мобильного приложения**

Intent используется для запуска компонентов Android и передачи данных. Явные интенты — указание конкретного компонента. Неявные — с указанием действия и фильтров. Применяются для запуска активностей, сервисов, передачи данных.

**Вопрос 48. Примитивы синхронизации: Семафоры**

Семафор ограничивает число потоков, одновременно использующих ресурс. Имеет счетчик, уменьшаемый при входе и увеличиваемый при выходе. Поддерживается в Java (Semaphore), Python (threading.Semaphore).

**Вопрос 49. Способы записи алгоритмов**

* Псевдокод — упрощенный синтаксис
* Блок-схемы — графическое представление
* Текстовое описание — понятный язык
* Код — реализация на языке программирования Использование зависит от цели: разработка, документация, обучение.

**Вопрос 50. Концепция виртуальной памяти**

Позволяет программам использовать больше памяти, чем доступно физически. ОС отображает виртуальные адреса в физические через таблицу страниц. Реализует защиту памяти и изоляцию процессов.

**Вопрос 51. ООП: Методы**

Методы — функции, определенные в классе. Делятся на:

* Обычные (экземпляра)
* Статические (не используют this)
* Абстрактные (реализуются в наследниках)
* Переопределенные (override) Методы обеспечивают поведение объектов.

**Вопрос 52. Ввод-вывод высокого уровня на консоль**

В Java — классы Scanner, BufferedReader для ввода, System.out для вывода. В Python — input() и print(). Позволяют работать с текстовыми данными, преобразовывать типы, обрабатывать ошибки ввода.